

AD-A117 450 RENSSELAER POLYTECHNIC INST TROY NY IMAGE PROCESSING LAB F/6 9/2
OCTREE GENERATION, ANALYSIS AND MANIPULATION.(U)

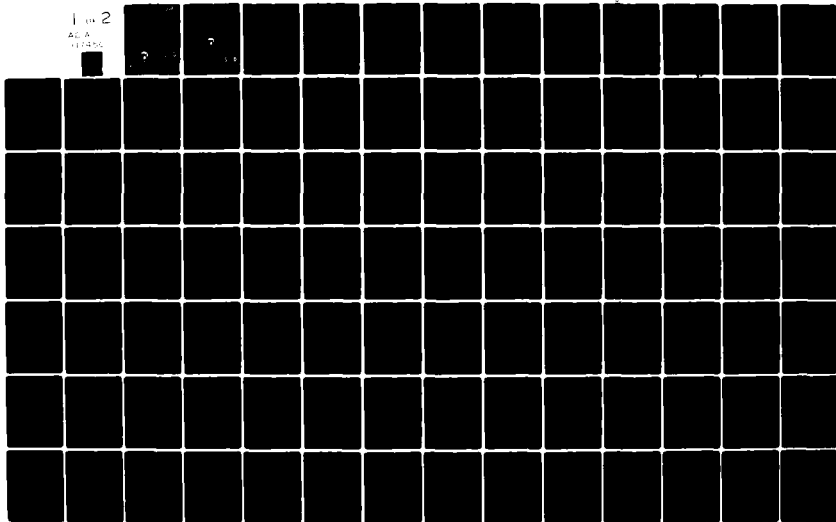
UNCLASSIFIED APR 82 D MEAGHER
IPL-TR-027

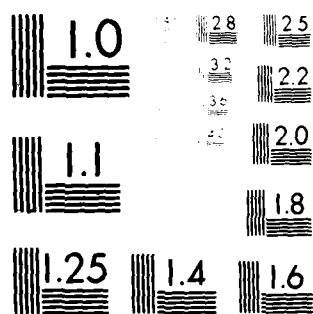
N00014-82-K-0301
NL

1 2

AC A

11/1/82





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A117450

12

DTIC FILE COPY



DTIC
ELECTE
JUL 23 1982
S D

Image Processing Laboratory

Rensselaer Polytechnic Institute
Troy, New York 12181

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

88 06 88 113

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By <u>Rec Ltr. on File</u>	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



IPL-TR-027

Octree Generation, Analysis
and Manipulation

Donald Meagher

April 1982



DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

IMAGE PROCESSING LABORATORY

Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute
Troy, New York 12181

DTIC
ELECTE
JUL 23 1982
S D D

ABSTRACT

Octree Encoding is a solid modeling method designed for the high-speed manipulation, analysis and display of arbitrary 3-D objects. The technique is based on a hierarchical 8-ary tree or "octree" for object representation.

Octree Encoding is presented and analyzed along with a discussion of the major considerations involved in its development. Techniques for the efficient conversion into octrees of convex polyhedra and restricted analytic objects are presented. Strategies for unrestricted and concave object conversion are also discussed. Algorithms for the measurement of object properties (volume, surface area, center of mass, moment of inertia, segmentation of disjoint parts, number of interior voids and a correlation between two objects), geometric operations (translation, scaling, rotation, concatenated geometric operations, nonlinear operations and perspective transformation) and rotational swept volume are developed.

ACKNOWLEDGMENT

The author would like to thank Professor Herbert Freeman of the Electrical, Computer, and Systems Engineering Department, at Rensselaer Polytechnic Institute for his guidance and support of this research.

Partial funding for this effort was received from the Office of Naval Research under Contract N00014-82-K-0301 and from Phoenix Data Systems, Albany, New York.

Table of Contents

1	INTRODUCTION.....	1
1.1	Existing Schemes.....	4
1.2	Goal.....	5
2	NEW APPROACH.....	7
3	DATA STRUCTURE AND PROPERTIES.....	16
3.1	Definitions.....	16
3.2	Object Representation.....	20
3.3	Node Requirements.....	24
3.4	Neighbors.....	26
3.5	Complexity Metric.....	28
3.6	Storage Requirements.....	29
4	TOOLS AND ALGORITHMS.....	32
4.1	Traversal Strategy.....	33
4.2	Object Generation.....	34
4.2.1	Orthogonal Blocks.....	40
4.2.2	Convex Objects.....	46
4.2.3	Object Generation Examples.....	68
4.2.4	Concave Objects.....	72
4.3	Object Properties.....	74
4.3.1	Volume.....	75
4.3.2	Surface Area.....	75
4.3.3	Center of Mass.....	76
4.3.4	Moment of Inertia.....	77
4.3.5	Segmentation of Disjoint Parts.....	78
4.3.6	Interior Voids.....	78
4.3.7	Correlation.....	79
4.4	Boolean Operations.....	79
4.5	Overlays.....	80
4.6	Geometric Operations.....	85
4.6.1	Translation.....	85
4.6.2	Scaling.....	87
4.6.3	Rotation.....	88
4.6.4	Concatenated Geometric Operations.....	91
4.6.5	Nonlinear Transformations.....	93
4.6.6	Perspective Transformation.....	96
4.7	Swept Volume.....	99
4.7.1	Rotational Swept Volume.....	99
5	REFERENCES.....	105

1 INTRODUCTION

The ability to model 3-D objects accurately and efficiently within computerized systems has assumed greatly increased importance in recent years. Application areas include computer-aided design (CAD), computer-aided manufacturing (CAM), robotics, medical imaging, cinematographic modeling, object recognition, and so on.

Many factors are driving the need for these systems, including economic forces such as the need for increased productivity in order to reduce inflation, the shortage of and increasing cost of skilled labor, the dramatically increasing (and possible shortage of) energy, and the rapid improvement in the performance-to-cost ratio for computing hardware. In manufacturing and other areas, energy costs and government mandated improvements in safety, efficiency, and emissions are necessitating new and "smarter" designs within shorter development cycles. Often much more complex and sophisticated products are required, increasing the need for computerized aids to augment or replace traditional manual methods.

In spite of the need and numerous efforts undertaken, at the current level of development, advancement in many application areas is being impeded because effective and practical object representation schemes and associated algorithms for real-time manipulation, analysis and display

are not available. Briefly, generally used techniques have a limited range of applicability because of shortcomings in two major areas. First, representation capabilities are not sufficiently robust to easily handle the object complexities required in a realistic environment. Second, manipulation and display algorithms performing functions such as interference detection (two or more objects occupying the same space) and hidden surface removal (necessary for realistic display) require extremely large numbers of calculations in practical situations. They usually exhibit exponential growth (often quadratic unless special steps are taken) in the number and complexity of the objects. It is not believed that near-term progress in hardware technology will render such schemes practical for the vast majority of potential applications. Much more efficient data structures and algorithms are needed.

The basic function of any scheme for representing rigid solid objects or Solid Modeling System (SMS) is to define some form of symbol structure to represent solid objects. In addition, input facilities must be available for creating objects or converting them from application data structures, inquiry facilities must be available so that application algorithms can be developed to analyze and manipulate the modeled objects and, finally, a method to output objects is needed for display, storage, and for possible use by other systems.

The inquiry function is of particular importance. It is believed that SMSs being developed today will form the lower levels of advanced computerized systems of the future. The higher levels will incorporate sophisticated artificial intelligence functions combined with powerful analytical techniques, both drawing upon vast knowledge structures containing virtually all information needed to completely or almost completely automate tasks such as part design, tool path generation, process planning, optimization, etc. As such, these lower level facilities will be utilized with a very high duty cycle and, just as with lower level operating system procedures, must be made very fast and efficient. In addition, they must be extremely reliable and completely automatic. Human intervention to resolve ambiguities obviously could not be tolerated. In order to be generally useful, the scheme and associated algorithms must also be powerful. It should be able to process and analyze all possible objects that could be created and must not halt or require unbounded processing time as object complexity increases. The scheme must be simple (no or few special cases, for example), transportable to a variety of machines and environments, and should be general enough to be usable over a very large domain of applications without modification.

1.1 Existing Schemes

Most commercially available CAD systems do not employ a true SMS in that 3-D objects are not really modeled. They are essentially extensions of drafting techniques based on the use of edges to represent solids in projection. The determination of what is actually solid and where the interior and exterior of an object lies is left to human interpretation. Most cannot reliably remove hidden lines or generate sectional views automatically.

Rejecting such schemes, existing SMSs have been divided into the following six categories by Voelcker and Requicha [1,2]:

- (1) Primitive Instancing - families of objects are defined parametrically. A shape type and a limited set of parameter values specifies an object.
- (2) Spatial Enumeration - an object is represented by the cubical spatial cells (volume elements or "voxels") which it occupies.
- (3) Cell Decomposition - a generalized form of spatial enumeration in which the disjoint cells are not necessarily cubical or even identical.
- (4) Constructive Solid Geometry (CSG) - objects are represented as collections of primitive solids (cuboids, cylinders, etc.). A tree structure is typically used with branch nodes representing Boolean operations and

leaf nodes the primitives.

(5) Sweep Representation - a solid is defined as the volume swept by a 2-D or 3-D shape as it is translated along a curve.

(6) Boundary Representation (B-Rep) - objects are represented by their enclosing surfaces (planes, quadric surfaces, patches, etc.).

Specific advantages and disadvantages of each have been tabulated [3] along with a classification of 21 existing systems. Most use CSG (TIPS, PADL, SYNTHAVISION, etc.) or Boundary Representation (BUILD, CADD, EUKLID, ROMULUS, etc.). In a separate study, Baer, Eastman and Henrion [4] have analyzed and compared 11 popular systems.

1.2 Goal

The goal of this research has been to devise a new SMS and associated linear growth algorithms in which objects of arbitrary complexity can be encoded, manipulated, analyzed and displayed interactively in real-time or close to real-time in parallel, low-cost hardware. Attempts were made to develop and incorporate improvements over existing systems in as many of the current problem areas as possible.

A solid modeling scheme called Octree Encoding was developed [5-7]. The technique employs a hierarchical N-dimensional binary tree (or a (2^N) -ary tree) to represent

an N-dimensional object. For a 3-D object this is an 8-ary tree or "octree."

APPROACH

The ultimate goal of this effort has been the actual construction of a full-function, real-time (about 1/30 second response) solid modeling system to handle any number of arbitrarily complex objects while operating on relatively fast, low-cost hardware. Conventional solid modeling systems have assumed these characteristics to be mutually exclusive. Levels of performance many orders of magnitude better than allowed by existing techniques are needed. Obviously a new approach is required!

The remainder of this section is an ex post facto attempt to rationalize the 4 or 5 years of evolving reasoning and philosophy embodied in the Octree Encoding method. The intent is also to sketch the paradigm within which the scheme was conceived and nurtured.

The first and perhaps most important step in this journey was to reject any preconceived ideas about solid modeling. It was not accepted that CSG or B-Rep was obviously the best approach. The lack of high level publications on the subject (until recent years) helped in this regard.

In the absence of clear guidelines and direction, perhaps the key to conjuring up an approach to such a task is an immutable set of priorities. From beginning to end this effort has been faithful to the number one priority:

high-speed operation. For the first few years the actual usefulness of the method was open to question but there was never doubt the functions could be performed at an extremely high throughput utilizing modest hardware.

The second priority was robustness. This included both the ability to represent arbitrary objects and a full complement of analysis, manipulation and display functions.

The third thrust was a general drive for simplicity. This required, for example, a single representation scheme for all objects so as to eliminate the quadratic growth in the number of interfaces between functions and primitives as functions and primitives are added to a system. From an implementation point of view, the benefits of simplicity were most aptly expressed by Gorden Bell et al. [8] of DEC:

"The cheapest and fastest parts of a system are those that don't exist."

Given these general priorities and preliminary thinking, a few more strokes in the outline can be sketched in. If arbitrary objects will be handled (at high-speed, no less!) the obvious problem must be faced. Arbitrary objects require arbitrary quantities of storage for representation. They can easily require infinite storage if all detail is to be preserved. The approach adopted here is to represent arbitrary objects to a variable but limited precision. Part of the philosophical justification for such a decision has been expressed from a pragmatic viewpoint by Daniel McCracken

[9] (although in a different context):

"I suspect that the more precisely you measure something, the less important it really is."

CSG schemes handle this dilemma by taking what can be looked at as the opposite choice. Infinite resolution (for all practical purposes) is preserved but objects are limited to primitive analytic shapes or combinations thereof.

The next step is to squarely face the computational complexity issue. Most existing CAD systems have evolved from attempts to automate the 2-D drafting function in engineering. Complexity has not generally been a consideration because the typical drafting task, such as generating a fillet, is linear in a very small number of items. Interactive operation is not difficult to achieve.

The progress of CAD into full 3-D applications has changed the situation. Operations which involve some form of interference detection such as collision avoidance and hidden-surface removal have been found to require very large, often prohibitively large, computational resources. The reason for this has not always been well understood or appreciated.

The root of the problem is a comparison task. Naive algorithms perform an interference detection operation by checking for intersection between each possibly relevant pair of primitives. A combinatorial explosion results because, in general, the number of pairs grows quadratically in the

number of primitives.

Many schemes have been proposed and developed to alleviate this problem. Most involve some form of spatial sorting of object or display space before interference detection is performed. The sorting phase is typically an $O(n \log(n))$ growth operation except when a radix sort can be performed such as is used in the typical Z-buffer approach to hidden surface removal. This is, of course, a vast improvement over quadratic algorithms but most such schemes have severe limitations in other areas as a result.

Two additional interference analysis characteristics should be considered. First, some schemes require a new sorting before each interference operation. Sorting for hidden surface removal before each scene generation is an example. A single pre-sorting for all time is preferred.

Second, some algorithms require all object elements to be examined for each interference operation. Z-buffer hidden surface removal is an example. In some cases, algorithm growth may be linear, but it is linear in the number and complexity of all objects. A preferred data structure would require only relevant sections of an object to be evaluated based on a single pre-sorting for all time. Such a data structure was developed for this effort.

The philosophical approach adopted for algorithm development was based on the hierarchical ideas of Clark [10] and the sharing of partial calculations that has been proved

so successful in the Fast-Fourier Transform (FFT). It is the hierarchical structure into which the problem has been cast that allows large numbers of low-level calculations to be eliminated at an early stage when processing typical objects.

The approach to actual implementation adopted for this effort was based on the current trends in VLSI technology. It is clear that to maximize the performance-to-cost ratio, full advantage should be taken of the tremendous improvements in hardware which have resulted and will no doubt continue to result from VLSI.

Before proceeding, a popular misconception concerning increased computing power should be dispelled. The thought that increasingly powerful hardware at lower cost will allow inefficient algorithms to become practical is, in general, wrong. In reality, a great increase will further widen the performance gap between an inefficient (quadratic growth, for example) algorithm and an efficient (linear) algorithm. Thus, computational complexity issues become more, not less important as technology moves into the VLSI age.

A new trend has been emerging for the implementation of high-performance systems in recent years. It combines advanced CAD tools and semi-custom integrated circuits. Increasingly, the designer will begin a system development by entering a register level block diagram into a CAD graphics workstation. After a timing analysis to detect problems such as race conditions he will be able to simulate system

operation. When satisfied, the circuit will automatically be reduced and a final layer or two of metalization for a predefined semi-custom chip will be generated (probably interactively). A working chip will be available a few days or weeks later.

As this is written, all of these tools are available at varying levels of usefulness. Semi-custom chips containing up to 10,000 gate equivalents (2 input NAND) are currently available. Full custom chips, microcomputers and memory chips are commonly available at the 60,000 to 70,000 transistor per chip level. Experimental units of up to almost 500,000 transistors have been fabricated.

The implementation approach is thus to develop algorithms designed specifically for semi-custom or full custom VLSI based operation. This decision impacts the entire design philosophy. Traditional solid modeling systems tend to be huge, ever growing, ever changing, software based packages with complex internal structures. The VLSI based system, in contrast, must be based on a small number of very simple, fixed, powerful and extremely reliable algorithms. They are implemented in hardware and form the primitive lower level functions in an applied system. Each could be viewed as a specialized peripheral processor in a conventional architecture.

During algorithm design, maximum advantage was taken of the more or less standard techniques that have proved to

enhance performance in hardwired digital systems. This includes extensive use of parallelism and pipelining, avoidance of iteration, looping or unbounded situations, balancing of gate delays, asynchronous bus designs, and so on.

The classical computation vs. memory tradeoff was generally decided in favor of extensive use of memory (including virtual) in order to increase performance. It is believed that in the near future the cost of real memory in most computerized systems will cease to be a major part of overall cost (one million bytes of semiconductor memory can be purchased for \$3600 at the board level or \$640 at the chip level as this is written with Brannon of Intel predicting [Computer Decisions, Feb. 1982, p. 12] a price of \$50 to \$100 by the end of the 1980s). In an advanced implementation, a 16M byte real memory, 1G byte secondary memory and 1T byte virtual addressing space is not considered unreasonable.

The algorithms were designed with what has recently been popularized as the "reduced capability machine" concept. Briefly, this holds that higher performance may be achieved by redesigning a processor with fewer functions (instructions) which execute at a much faster rate. The larger number of instructions required to be executed in the performance of a task will, hopefully, execute in a shorter time period.

Based on this approach, none of the basic algorithms or resulting processors were allowed to employ floating-point numbers or operations. In addition, neither integer multiplication nor integer division (other than integer shifts) were allowed. Legal operations are integer addition and subtraction, magnitude comparison, shifts, and data movements such as LOAD, STORE, stack PUSH, stack POP, queue INSERT, and queue DELETE. These legal operations form a set that will be called simple arithmetic. These are serious restrictions but result in faster and simpler processors, lower cost, shorter gate delays, etc. In hardware, for example, a fixed shift is essentially free in cost and processing time because output data lines can be simply hardwired into a shifted position. It should also be noted that the per unit cost of a processor may become very important when multiprocessing is being considered.

Two phases of algorithm operation are defined, setup and run. In the first phase, a full function general purpose processor such as a microprocessor accepts the user request. This could include, for example, view angles in floating-point degrees. The request is converted into a set of integer setup and control values that are then loaded into the appropriate registers in a second, specialized processor. This processor is a hardwired unit designed to perform a single SMS operation using only simple arithmetic. In an operational system many such processors would be employed.

In the second phase, the slower full function unit is essentially idle while the requested operation is performed on the objects at a very high rate by the specialized processor. After processing a few calculations may be allowed on the general purpose machine to place the results in user format.

Computers have often been viewed as number manipulators. In reality, they are simply symbol manipulators in which the manipulations typically correspond to mathematical functions. The processors envisioned here are also symbol processors. The symbols are elements of solid objects and the manipulations are desired solid modeling functions. They could be looked at as specialized "solid processors" or "solid engines."

3 DATA STRUCTURE AND PROPERTIES

3.1 Definitions

A graph $G(N,E)$ is a finite, nonempty collection N of nodes and a set E of unordered pairs of distinct nodes called edges. Two nodes connected by an edge are adjacent nodes. If an edge has an associated direction, it is a directed edge. The direction is from the tail node to the head node. A graph containing only directed edges is a directed graph or digraph. The number of edges which have a node as their tail node is the outdegree of that node. The number of edges which have a node as their head node is the indegree of that node. A graph which has no paths which originate and end in the same node is called acyclic.

A tree is an acyclic directed graph in which all nodes have indegree 1 except one node, the root, which has indegree 0. Any node which has outdegree 0 is called a terminal node or leaf. Nodes with outdegree greater than 0 are branch nodes. The level is defined as the distance in edges from the root. The root is at level 0.

The root is assumed to be at the top of the node structure and all other nodes exist below the root. All nodes reachable from a particular node are called the descendants of that node. All nodes from which a particular

node can be reached are the ancestors of that node. Descendants one level below a node are the children of that node. The original node is the parent of the child node.

If a node does not actually exist in a tree but can be inferred from an existing terminal node which would be (or will be) one of its ancestors, it is called an implied node. Loosely, operations on a tree which use implied nodes are said to process the implied tree rather than the actual tree.

Every branch node is a root of one or more subtrees. The subtrees immediately below a node form a set of disjoint trees called a forest. The degree of a node is the number of subtrees that exist for that node. If the outdegree of every branch node is $\leq m$, the tree is an m-ary tree. If the outdegree of every branch node is m , the tree is a complete m-ary tree. For a binary tree or a complete binary tree, for example, $m=2$.

A positional m-ary tree is an m-ary tree in which the children have m distinct positions. The position of a node is indicated by a value from the child number set $\{0, 1, 2, \dots, m-1\}$.

Every node is uniquely identified by a node address which is a string over the child number set. The root is represented by the empty string. The node address of a child is the child number prefixed by the address string of its parent.

A tree will be called a hierarchical tree if the

children of a node are associated with their parent in some particular relationship.

All objects exist within the universe. It is a finite section of N-dimensional space defined by N orthogonal axes and $0 \leq x(i) \leq e$ where $x(i)$ is a displacement in dimension i , $(x(1), x(2), \dots, x(N))$ is a point in the universe, e is the length of an edge of the universe and N is the order of the universe (number of dimensions). The character "N" will be reserved for this purpose throughout the report.

Note that all edges of the universe have the same length forming a square for $N=2$, a cube for $N=3$ and an N-dimensional hypercube for $N>3$. The origin of the universe is the point of intersection of the axes. Negative displacements from the origin are not allowed. The space beyond the universe is the void. No object can exist in the void. Any part of an object moved into the void is annihilated. An augmented universe is one in which one or more adjacent (empty) universes are added to the primary universe. Augmented universes are used to facilitate algorithm initialization.

Before encoding, objects are called real objects. They may be real world objects or a mathematical description of an ideal shape. An object encoded in the Octree Encoding format is known as the encoded object or simply the object.

If a single encoded object is used many times to generate new, transformed objects, the original object is the model and the new objects are instances.

An object can have any number of dimensions. A one-dimensional object is one or more segments of the axis forming a one-dimensional universe. A 2-D object occupies area, a 3-D object occupies volume, a 4-D object can be thought of as occupying spacetime, and so on.

An object is always of the same order as the universe in which it is defined and is composed of discrete units of N-dimensional space. All objects in a third order universe must occupy volume, for example. A 2-D object could not exist here. The smallest object in such a universe would be the smallest resolvable unit of space. No object can occupy a point (zero volume).

Other than this, there are almost no restrictions on objects. They can be concave as well as convex, have any number of interior voids, and can be composed of multiple disjoint parts.

Each object is defined over the entire universe. It has a property value defined at each point in the universe. For a typical small object (relative to the universe) most of the space in the universe has the property of being empty.

An encoded object B is thus defined as a family of ordered pairs $B(k) = (P, E(k))$ where P is a finite set of properties and E(k) is the set of disjoint object elements or obels which exactly fill the universe at level of resolution k in a manner described below. Obels are distinguished from spatial enumeration voxels because they are not uniform in

size and not necessarily three-dimensional.

In Octree Encoding, the obels which constitute an object are represented by the nodes in a tree structure. The tree contains all members of the family of objects of increasing resolution up to some maximum level of resolution. The obel in space and the node in a tree structure representing it are generally considered to be synonymous throughout this report.

3.2 Object Representation

Take a positional m -ary complete tree in which each node represents a section of N -dimensional space. Divide the section in each dimension represented by each parent into p equal disjoint segments numbered 0 to $p-1$. This forms p^N sections of N -dimensional space $j(i)(e/p) \leq x(i) \leq (j(i)+1)(e/p)$, $j(i)=0,1,2,\dots,p-1$ & $i=1,2,\dots,N$ where i is the dimension, x is a displacement from the origin of the parent and e is the length of an edge of the parent.

The sections of space are the obels of an object and are represented by the $m=p^N$ children of the parent. The root of the tree represents the universe and contains all N -dimensional space. The level of the obel is the level of the corresponding tree.

The child number is defined to be:

$$\text{child number} = \sum_{i=1}^N (p^{(i-1)} s(i)) \quad (3-1)$$

ere $s(k)$ is the segment number in the k -th dimension.
ld number set is thus $\{0, 1, 2, \dots, p^N - 1\}$.

or the remainder of this report, only a value of $p=2$
is considered. A binary complete tree in each dimension
herefore be used forming an N -dimensional binary tree
ditional m -ary complete tree in which $m=2^N$.

hus, a one-dimensional binary tree is simply a binary
A 2-D binary tree is a quadtree. A 3-D tree has 8
en and will be called an octree. A 4-D tree has 16
en and is a hexadecatree.

These tree structures are hierarchical because the
obels represent the identical section of space as the
en. Thus, the trees as used here should not be
sed with the more or less standard tree structures used
ta processing to maintain items in a sorted format.
y to the identification problem is the fact that such
ierarchical structures have also been called "quad
" when representing data items with a double index.

This report will only consider terminal nodes which are
etely defined by the corresponding properties. In other
, the space occupied by a terminal node is homogeneous.
complex forms are possible but will not be discussed.

The binary object property will be used in most of this
t. A terminal node will be either FULL (an "F" node) if

the corresponding obel is completely occupied by the object or EMPTY (an "E" node) if the corresponding obel is completely disjoint from the object. Branch nodes are PARTIAL (a "P" node) meaning partially full. It is assumed that additional properties such as color or texture values, material type, function, density, surface normals, thermal conductivity, etc. are simply attached to F nodes. The algorithms below can be extended to process such trees if an operation is defined to combine properties for the various transformations.

When a real object is converted to octree format, branch nodes at the lowest level must be given a terminal value. This could be E or F if the obel is less than or greater than half occupied, respectively. If interference detection or collision avoidance is needed, the worst case situation is usually assumed; they are given the value F.

If all of the children of a branch node are terminal with the same property value, they are unnecessary and should be eliminated. The parent node is converted to a terminal node containing the child property. If a tree contains no such nodes it is a reduced or trimmed tree.

Unreduced trees are sometimes created during algorithm operation. They are valid and are correctly processed by most algorithms but cause inefficiency. Versions of algorithms that generate trees in a depth-first sequence typically eliminate unnecessary nodes during operation. The

output is a reduced tree. Otherwise a separate reduction pass through the tree is usually performed.

In specific applications an additional terminal node type, a tolerance node, is used. In a material removal operation, for example, they represent the tolerance object. This is the space within the required tolerance of the surface of the minimum desired object which can be optionally removed. Tolerance nodes are handled in a special manner by processing algorithms. They can be used as E nodes to obtain the minimum object, as F nodes for the maximum object or locally converted to whichever would result in the greater node reduction for the minimum storage object (assuming tolerance information no longer needed).

It should be noted that the location of any obel in the universe is known very accurately. In fact, it is known exactly. Likewise, the distance between any two obels in the universe is also precisely known. They are at fixed locations in space. The ambiguity existing as a function of tree level is in the location of the surface of the object within an obel. This distinction can be important, for example, when two parts of an object must be in very accurate alignment over a great distance. It is not necessary to maintain a precise representation between the parts.

Figure 3.1 is an example of a four level quadtree. Note child and vertex labeling conventions. They are selected to facilitate extraction of dimensional information based on bit

position. A three level octree object is shown in Figure 3.2.

The node address identifies a particular node and also locates the section of space represented by the node.

A node address in a 1-D tree is a binary string. The number of bits is equal to the level of the node. The value is the number of the section of the 1-D universe occupied by the node, numbered from 0 at the origin to $2^n - 1$ where n is the level. In 2-D, the address will be a string of single digit quaternary numbers and in 3-D, octal numbers. The section of a higher order universe can be likewise determined by independently considering the individual bit for each dimension in the child number values. On occasion a node is identified by its level and the decimal equivalent of the binary value of its address string.

3.3 Node Requirements

The actual number of nodes required to represent an object is a function of the size and shape of the object, its position and orientation when digitized, the level of resolution, etc. Of particular significance is the result that the surface area of a 3-D object sets an upper bound on the memory required.

Assertion 3.1: For a 3-D connected object, the number of nodes required for octree representation is on the order

of the product of the surface area of the object and the inverse of the square of the resolution.

Proof: A function $g(n)$ is defined to be on the order of $f(n)$ or $O(f(n))$ if there exists a constant c such that $g(n) \leq cf(n)$ for all but some finite (possibly empty) set of non-negative values for n .

Consider a 3-dimensional universe defined to level n . Without loss of generality, the volume of the universe is defined to be 1. Resolution at level n will be defined as the edge size, e , of an obel at level n or $e=1/2^n$. The resolution of the object, r , is the edge size at the lowest level or $r=1/2^m$. Consider the minimum surface object which intersects 8 obels (any level) and continues on to touch a ninth. It intersects 8 obels at and around the common point at which all 8 touch and continues along the entire length of an edge. It will be a linear run of minimum level obels for a distance e and has a surface area of $4re+2r^2$. For an object to actually intersect all 9, it must be larger than this and have a larger surface area.

Let S be the surface of an object. Let k be the number of cubes at level n which could be required to represent the object.

In a worst case situation, the surface area would cover a maximum length run of minimum level obels, which sets a limit on the number of obels at level n which can be intersected:

$$k < 8(S/(4re+2r^2)+1)$$

The 1 accounts for (actually, more than accounts for) the four obels which could be intersected along with obel 9 at the far end of the run.

$$k < 8(S/4re+1) = 2S/re+8$$

Let C be the total number of obels or nodes required to represent an object:

$$C < \sum_{n=0}^m (2S/re+8)$$

$$C < (2S/r) \sum_{n=0}^m (2^n) + 8 \sum_{n=0}^m (1)$$

$$C < (2S/r)(2^{m+1}-1)+8(m+1) = (2S/r)2^m - 2S/r + 8m + 8$$

$$C < 4Sr^{-2} - 2Sr^{-1} + 8m + 8 \text{ or } C \text{ is } O(Sr^{-2})$$

Q.E.D.

3.4 Neighbors

The points in and on an obel are defined by a range of values in each dimension. A restricted set of points is defined if the coordinate values in one or more dimensions are fixed. Types of restricted sets are defined by the number of dimensions restricted for an N-dimensional obel as follows:

<u>Dimensions</u>	<u>Restricted</u>	<u>Type</u>	<u>Dimensionality</u>
N>=1	N	Vertex	0 (point)
N>=2	N-1	Edge	1 (line)
N>=3	N-2	Face	2 (plane)
N>=4	N-3	Prime	3

The fourth type, prime, is only defined for a 4-D universe and contains 3-D space. It could be a 3-D object at a specific time or two spatial dimensions and time restricted in the third spatial dimension.

For each dimension there are two values on the surface of the obel. The number of surface subsets of each type is the product of the number of cases of restricted values (2^n where n is the number of restricted dimensions) and the number of combinations of n out of N dimensions ($N!/(n!(N-n)!)$). They are tabulated as follows:

<u>n</u>	<u>1-D</u>	<u>2-D</u>	<u>3-D</u>	<u>4-D</u>	<u>Type</u>
N	2	4	8	16	Vertex
N-1		4	12	32	Edge
N-2			6	24	Face
N-3				8	Prime
	---	---	---	---	
Totals	2	8	26	80	

The total number of such subsets is listed. It is 3^{N-1} or the number of possible cases in each dimension (two fixed values plus the "not restricted" case) raised to the number of dimensions less the one case of no restriction in any dimension.

Neighbors are defined as obels that share one or more surface subsets. They are vertex, edge, face or prime neighbors depending on the shared subset. Note that the

types are inclusive as the number of shared dimensions increases. For example, an edge neighbor is also a vertex neighbor and a face neighbor is also an edge and vertex neighbor.

3.5 Complexity Metric

An important item that is generally lacking in the field of 3-D solid modeling is a measure of object complexity. It is difficult to study a situation analytically when quantitative measures are not available. Intuitively, the measure of the complexity of an object should in some sense be related to the amount of information required to represent the object.

The measure of object complexity used here is the number of nodes in its octree [6]. The actual complexity value will depend on location and orientation. If needed, objects could be normalized by translation so that they touch the three sides of the universe that intersect the origin. If there is no preferred object orientation, an average could be calculated for the object at uniformly distributed rotation angles. Alternately, the minimum (or maximum) number of nodes could be used.

An advantage is the ease of calculation. A complexity value can be generated automatically for any object with existing algorithms without manual intervention.

In the remainder of this report the letter "C" will be reserved to represent the node count (any type) in an object's reduced tree. The value of C is the sum of the number of branch nodes, B, and leaf nodes, L. The following relationships hold:

$$C = B + L = B(2^N) + 1 \quad (3-2)$$

$$B = (C-1)2^{-N} = \lfloor (2^{-N})C \rfloor \quad (3-3)$$

$$L = C - (C-1)2^{-N} = (1-2^{-N})C + 2^{-N} = \lceil (1-2^{-N})C \rceil \quad (3-4)$$

A tabulation by N is:

	<u>1-D</u>	<u>2-D</u>	<u>3-D</u>	<u>4-D</u>
B =	$\lfloor C/2 \rfloor$	$\lfloor C/4 \rfloor$	$\lfloor C/8 \rfloor$	$\lfloor C/16 \rfloor$
L =	$\lceil C/2 \rceil$	$\lceil 3C/4 \rceil$	$\lceil 7C/8 \rceil$	$\lceil 15C/16 \rceil$

Within the count of leaf nodes, the number of nodes with a value of E (or F) can range from 1 to L-1.

3.6 Storage Requirements

A minimum usable scheme requires two types of data items per node, a property value and pointers to its children (if a branch node). Additional data items which could be used are parent pointers, multiple property values, average subtree properties, sibling pointers, object feature pointers, pointers into application data structures, etc.

Normally a two bit field is used to encode the 3 node

values (E, F and P) requiring $2C$ bits or $C/4$ bytes for an object. This can be reduced by allowing for a single bit value. For example, a 0 bit could represent an E node while a 1 bit indicates the presence of a second bit to distinguish between F and P nodes. Since branch nodes will always be less than one quarter of the nodes for $N \geq 2$, a 2 bit code is used for P nodes. The choice of value for single bit encoding can be fixed or could be selected to minimize storage use. If fixed, 3-D storage will range from about $9C/8$ bits ($0.14C$ bytes) to about $2C$ bits ($0.25C$ bytes). When the optimum 3-D correspondance is selected, the worst case is reduced to about $25C/16$ bits ($0.2C$ bytes). This corresponds to a savings of approximately 20% minimum to 44% maximum. More sophisticated encodings based on higher order statistics are possible but will not be discussed.

Conceptually, each branch node of an octree has 8 storage fields for child pointers when stored in linked format. At implementation time, however, a single location will suffice because it is a complete tree. The children can be located in blocks of 8. A single pointer to the block together with an offset (0 to 7) will uniquely locate each child. In practice a single word of storage (typically 32 bits) per node can hold both the value field and pointer field.

Memory requirements can be substantially reduced if node storage is sequentially allocated. Using a heap-like storage

format, the position of a value in a string indicates its tree address. Two bits per node (or less) is sufficient. The allocation can be breadth-first or depth-first. The major disadvantage is that, similar to a magnetic tape, all earlier nodes must be read before a desired node can be located. It is considerably slower for selective depth searches.

Parent pointers are probably not necessary because in any real application the number of levels is limited. With a 32 level octree, for example, objects could be represented to a resolution of 0.001 inch in a universe enclosing 311,482.8 cubic miles. In such a situation, depth-first traversal algorithms could keep parent pointers in a small stack.

In some applications subtrees can be shared within an object or between objects. Pointers are simply allowed to point to the same node (root of the shared subtree). This may, however, greatly complicate object modification and deletion.

4 TOOLS AND ALGORITHMS

The intent of this section is to develop a body of "tools" to be employed by algorithms in the performance of sub-functions within the implementation constraints (simple arithmetic, easy VLSI implementation, etc.). This will be a "bag of tricks" from which specific solutions will be drawn as the need arises. These tools perform specific, not general, functions but will, hopefully, be broadly useful over many algorithms. Lower level tools will be combined to form higher level ones to implement more sophisticated functions. At the lowest level tools consist of the simple arithmetic operations. To the system implementor the tools will correspond to specific hardware subsystems to be used in the construction of special purpose hardware processors.

In order to motivate tool development, their introduction will be placed within the context of solutions to increasingly difficult modeling system functions. In most cases, solutions to the 2-D (or 1-D) problem will be presented first for clarity, followed by the extension to 3-D.

4.1 Traversal Strategy

In general, implementation details will be avoided. It should be noted, however, that two categories of tree traversal sequences, depth-first and breadth-first, are possible. They correspond to two strategies for attacking problems. A depth-first algorithm generally traverses a tree downward from parent to child, returning to the parent when all lower nodes have been processed. Breadth-first traversal processes all nodes at one level before working at the next lower level.

Depth-first traversal tends to be used when local information is required while breadth-first is employed when global information is needed. For example, determining if a point is interior to an object requires local information in the vicinity of the point. A depth-first descent to the obel containing the point would be used. On the other hand, determining if a spatial interference exists between two objects (but not the actual intersection) would typically call for a breadth-first traversal in the hope that interference between two solid obels would quickly be detected (or determined not to occur) at a high level.

A depth-first algorithm is generally spatially oriented. It has advantages when an overall spatial ordering is needed such as display with hidden surfaces removed. A breadth-first algorithm is typically more object oriented.

The entire object is processed at increasing levels of fidelity. Many objects can be in process with results at a high level (low resolution) becoming available before the lower level detail of objects are even accessed in memory.

In an application, tree traversal is not necessarily restricted to either depth-first or breadth-first but system performance may be optimized by allocating memory according to the predominant traversal strategy. Depth-first operations typically use a stack either directly or via reentrant code to maintain tree location. Breadth-first information is passed from one level to the next in a queue. The depth of the stack is usually quite modest while the length of the queue can be very great at the lower levels of complex objects if few high level nodes have been eliminated from consideration.

4.2 Object Generation

One of the first tasks to be addressed when considering an implementation of a SMS based on Octree Encoding is object generation or more specifically, conversion from application representations such as CSG and B-rep into the octree form. The user entry of a string of node values is obviously not an effective entry scheme. It is expected that the high-speed conversion from various high level application formats will be required.

Perhaps the most obvious method is the brute force use of spatial enumeration. A full tree is first constructed with all possible leaf nodes at the desired lowest level. The input objects are processed with the leaf nodes corresponding to the interior voxels marked F. As noted by Pichha [2] conversion from any popular SMS format to spatial enumeration is straightforward. Properties could be attached at this time. The tree is then simply pruned.

For a full tree stored sequentially, the storage address for the level n node at location (X_1, X_2, \dots, X_N) where each dimension coordinate ranges from 0 to $2^n - 1$ will be given by:

$$\text{offset} = \sum_{i=0}^{n-1} (2^{Ni} + \sum_{j=1}^N (2^{Ni+j-1} \text{BIT}(X_j, i))) \quad (4-1)$$

where $\text{BIT}(X, k)$ is the value (0 or 1) of bit location k (the least significant bit is location 0) of variable X .

The obvious problem for most non-trivial cases is the memory requirement for spatial enumeration storage (2^n where n is the lowest level) and the associated processing time (all leaves must be accessed at least once). Caching in a virtual memory environment could be a serious problem, depending on the spatial randomness of solid voxel enumeration during object conversion. To conserve memory, multiple passes through the object could be performed for divisions of the universe, followed by union operations.

A variation is to generate complete 2-D quadtrees representing orthogonal slices through the universe. The corresponding planes through the input objects are then examined. Interior squares are marked F. The result is essentially a binary image of a slice through the universe. A simple algorithm then converts the quadtree to an octree (voxels on a plane). Multiple slices are then unioned. All possible obels in the universe are still accessed but the memory required may be substantially reduced. This method was used to generate the skull octree objects from 2-D CT images as presented below.

For most applications, an object generation algorithm which requires the processing of all possible minimum level voxels in the universe for each object is unacceptable. In addition to the expense in time and memory, it runs counter to a general strategy of converting from application format to octree format in a top-down manner on a demand basis. More efficient algorithms are needed.

One could store octree representations for generic shapes to a very high precision in virtual space and perform linear transformations upon them at high speed. In situations such as tolerance checking a very high precision representation of a small part of the object surface may be required. The storage required to maintain such precision over the entire surface of all objects could be prohibitive.

A proposed solution is the use of specialized software

or hardware processors called octree generators. As shown in Figure 4.1 they would be preloaded with the object parameters. The user of the data, the octree processor, would request node values. The generator maintains the state of the traversal in an associated stack or queue.

In order to allow random traversal of the tree, the request could specify a particular connected node relative to the current node. For a depth-first traversal, a request would specify the parent or a child of the current node. For breadth-first, in addition to a request for the next node, it would be useful to allow a request to delete a current P node if it and its subtree were found to be unneeded. Its token is simply not inserted into the queue. At lower levels, the subtree would be unknown and therefore not generated.

From a complexity viewpoint, the efficiency of an octree generator is a function of the false-P rate. This is the fraction of nodes marked P that will have a value of E or F after reduction. This is not considered to be an error because the obel has not been incorrectly determined. The calculation of the final value for the node has simply been postponed, requiring additional work.

If the false-P rate is zero, all nodes are correctly determined the first time and the tree as generated is identical to the reduced tree. If the obel values can be determined in constant time, the computations grow linearly with object complexity (C). If, for a typical user request,

only parts of a tree are needed, computations could be expected to be linear (in some sense) in the complexity of the specific case.

Conceptually, an unsorted input object can be converted into a sorted octree in linear time, rather than $O(n \log(n))$ or worse time because a radix type sort over a finite "alphabet" (the obel locations) is involved. The worse than linear growth of typical sorting operations is caused when comparisons between elements is required. None are required here.

Something of a worst case example of a high false-P rate (approaching 1.0) algorithm would be the subdivision of all obels to the lowest level before status determination. The above spatial enumeration conversion to octree is an example. Since the computations are on the order of the volume of the universe, $8^{(\text{levels}-1)}$ this is obviously unacceptable in most situations. Thus, a zero or very low false-P rate (ie., the correct determination of final node value in all or almost all cases) is highly desirable.

The basic octree generation operation is to compare a test obel and the object being converted. The result is one of the 3 status values, E if $\text{Obel} \cap \text{Object} = \emptyset$, F if $\text{Obel} \cap \text{Object} = \text{Obel}$ or P otherwise.

The octree generation strategy is as follows. Beginning with the root node the values of test obels in the output octree object are determined by comparison to the input

(real) object. A node in the octree is created with this value. In most situations of interest this can be performed in constant time. E and F nodes are terminal and no longer considered. P nodes are subdivided with the corresponding children used as later test obels.

The first algorithm tool to be developed is the calculation of child vertex coordinates from the parent values. As shown in Figure 4.2 for 2-D, it is easily accomplished with additions and shifts (divide by 2). The 3-D or N-D equivalent is obvious.

For depth-first generation, subdivision is performed immediately with parent information saved in a stack. For breadth-first, all P nodes at a level are tested as they are deleted from the queue. Partial children are inserted into the queue for subdivision at the next lower level.

It may be convenient to consider the input objects as existing in an infinite "super universe" which encloses the octree universe. In this view, real objects can be infinite in length or enclose infinite space or exist beyond or outside of the octree universe before conversion. Thus, it is entirely legitimate for an input object to generate a single empty or full node if it is disjoint from or encloses the octree universe, respectively.

4.2.1 Orthogonal Blocks

The development of object generation tools will begin with a very simple case, a 3-D block (rectangular parallelepiped) with faces perpendicular to the major axes. The algorithm could also be used in 2-D for orthogonal rectangles or on 4-D (or higher) "hyperblocks." The basic tools developed here are very simple, but provide a foundation and framework for later tools. They will be applied repeatedly to more advanced functions.

A typical case is illustrated in Figure 4.3. The minimum and maximum coordinates in each dimension are noted. Three test obels with the three status values are shown.

Because the bounding planes and axes are orthogonal, the status determination can be decomposed into an independent comparison in each dimension. The projections of the test obel and object on an axis is shown in Figure 4.4. The projections are completely defined by their minimum and maximum values.

The comparison of the two projections can be analyzed by first noting that two coordinates and two projections results in four comparisons, Obel_Min to Object_Max, Obel_Max to Object_Min, Obel_Max to Object_Max, and Obel_Min to Object_Min. Each magnitude comparison can result in one of 3 determinations; the first value is less than, equal to or greater than the second. There are 3^4 or 81 combinations.

Two constraints reduce the possible combinations. First, the obel occupies space. Obel_Min must therefore be less than Obel_Max. Second, if the object is required to occupy space in at least the "super universe" Object_Min must be less than Object_Max. The 13 possible situations which remain are tabulated and illustrated in Figure 4.5.

If no section of the axis is contained in both projections, they are disjoint. A single point is not considered a section of the axis. If a single point is shared, the projections touch. Such cases are disjoint but are separately defined for possible future use.

If the projection of the obel is completely contained in the object projection, it is covered. If a section is contained in both but the obel is not covered, it will be called an overlap case. The three cases, disjoint, covered and overlap are defined to be mutually exclusive. Note that certain situations only occur when the length of the projections are restricted.

The various situations can be catagorized with four or fewer magnitude comparisons (for each dimension) as follows:

```
if OBEL_MAX<=OBJECT_MIN or OBEL_MIN>=OBJECT_MAX
then CASE<-'DISJOINT' else
if OBEL_MIN>=OBJECT_MIN and OBEL_MAX<=OBJECT_MAX
then CASE<-'COVERED' else CASE<-'OVERLAP'
```

In some restricted situations, not all comparisons need be performed. For example, if the obel projection is larger than that of the object, it cannot be covered. Overlap has

been determined after the first set of two comparisons.

The status of the obel can now be determined as a function of the projection cases.

Assertion 4.1: If the test obel projection and object projection are disjoint in one or more dimensions, the obel and the object do not intersect.

Proof: Select a dimension in which the projections are disjoint. For a non-touching situation, there is no coordinate value in this dimension contained in the projections of both the obel and the object. But any point of intersection must have a coordinate in this dimension and it must be in both projections. Therefore, no such intersection point can exist.

For a situation in which the projections touch, only a single coordinate value in this dimension is common. Therefore, the space occupied by the intersection will have a dimensionality at least one less than that of the universe, a situation of no intersection, by definition. Q.E.D.

Assertion 4.2: If, for all dimensions, the projections of the test obel and object overlap or the test obel projection is covered, the obel and the object intersect.

Proof: An algorithm is sufficient. For each dimension, select the minimum and maximum coordinate values which reside in both projections. Taken over all dimensions, they specify a section of space of the same dimensionality as the universe residing in both the test obel and the object. Q.E.D.

Assertion 4.3: If, in at least one dimension, the test obel projection is not covered by the object projection the test obel is not enclosed.

Proof: Select a dimension in which the test obel projection is not covered by the object projection. Select a coordinate which is in the test obel projection but not in the object projection. In the other dimensions, select any coordinate within the test obel projection. The coordinates define a point which is in the test obel but not in the object. The test obel is therefore not enclosed by the object ($\text{Obel} \cap \text{Object} \neq \text{Obel}$). Q.E.D.

Assertion 4.4: If, for all dimensions, the projection of the object covers the projection of the test obel, the test obel is enclosed by the object.

Proof: The coordinates of any point in the test obel are clearly also the coordinates of a point in the object. The test obel is therefore enclosed ($\text{Obel} \cap \text{Object} = \text{Obel}$). Q.E.D.

A determination of node status as a function of the set of cases for the individual dimensions is now possible. If the obel and object are disjoint in any dimension, there is no intersection and the node is marked E. If the obel is covered by the object in all dimensions, the obel is covered by the object. The node is marked F. Otherwise, there is either overlap or covering in all dimensions with at least one being overlap. The obel intersects but is not enclosed

by the object, generating a P node.

This is summarized as follows:

```
if CASE = 'DISJOINT' in any dimension
then STATUS <- 'E' else
if CASE = 'COVERED' in all dimensions
then STATUS <- 'F' else STATUS <- 'P'
```

All of the tools necessary to generate an octree for a block are now available. Beginning with the root, the nodes are generated by determining the status values and continuing the ambiguous ones (P nodes) to lower levels. The minimum and maximum coordinates in each dimension of the children are calculated from the parent nodes and compared to the coordinates of the block (fixed values for a specific block). The node status value is generated as a predicate combination of the results of the individual dimensional comparisons.

Since there are no cases in which an empty or enclosed node will be given a P value, the false-P rate will be 0. The reduced tree is generated.

A gage of performance can be made by determining the number of arithmetic operations required. An add/shift operation is used to generate child vertices and a magnitude comparison for status determination.

One add/shift is required for each dimension for each obel subdivision. It is not possible to eliminate the calculation in any dimension. All are required. This is shown by noting that no branch node will generate all E valued children. It would be an unreduced E node itself,

indicating a false-P case. Thus a P or F node must be among the children, requiring, as can be noted from the above algorithm, a non-disjoint (covered or overlap) case in all dimensions. This, in turn requires that all minimum and maximum obel vertex coordinate values be tested for all dimensions. A final observation is that either the minimum or maximum value is the parent average generated by the add/shift operation. Thus, N are necessary for each branch node requiring $(C-1)(N/2^N)$ operations.

The number of magnitude comparisons necessary depends on the specific object. A minimum and maximum will be determined. For each dimension one or two operations are required for a disjoint case and four for overlap or covered. For a final status value of P or F, an overlap or covered case is required for all dimensions, or $4N$ comparisons. For a status value of E, however, a disjoint case in a single dimension is sufficient, requiring a minimum of one operation. In any case, less than $4N$ will be required.

Thus, a minimum of C magnitude comparisons are required (a single node tree with a root value of E) with a maximum of $4NC$ (a single node tree with a root value of F).

4.2.2 Convex Objects

The generation of convex objects is more difficult than blocks because the dimensions cannot, in general, be analyzed independently. The simplicity and efficiency of the techniques of the last section will be used, however, with an orthogonal "bounding box" enclosing the object to perform a quick and inexpensive elimination of test obels clearly outside the vicinity of the object.

The basic low level operation in convex object generation is determining if a particular vertex point of a test obel is interior or exterior to the object. If the object is a convex polyhedron, the surface is described by a set of planar faces each of the form:

$$Ax + By + Cz + D = 0 \quad (4-2)$$

Each plane divides the universe into two half-spaces. The half-space containing the object will be called the positive half-space; the other is the negative half-space. The normal to the plane is defined to be in the direction of the negative half-space (away from the object). For points not on the plane (4-2) will evaluate to a positive or negative value depending on the half-space.

A simple substitution of the coordinates of the vertex point determines the location to be on the plane or in one of the half-spaces. Note that a positive evaluation value does

not necessarily indicate the positive half-space. Also, a point in the positive half-space is not necessarily within the object. The face plane is the entire plane while the face is a segment of the face plane.

If multiplications are not allowed, the components of (4-2) can be calculated for obel vertices by averaging from the parent components. A set of components for each vertex point in each dimension for each plane would need to be maintained. In 3-D this requires 24 values per plane.

Alternately, the signed perpendicular distance from the vertices of the universe to the plane can be calculated from:

$$d = (Ax+By+Cz+D)/((A^2+B^2+C^2)^{1/2}), \quad (4-3)$$

The distance value for a new vertex in a child obel is the average of the two parent vertices defining the edge containing the new vertex. The sign of the distance indicates the half-space. The new distances are calculated as follows:

$$(D_VAL(AND(CHILD,VERT))+D_VAL(OR(CHILD,VERT)))/2 \quad (4-4)$$

where CHILD is the child number of the new obel, VERT is the number of the desired vertex of the new obel and D_VAL(n) is the distance of parent vertex number n.

For each plane, 8 values are used. In practice, only one distance value need be kept. The remaining ones can be generated by adding an offset from a precomputed table.

From the definition of convex polyhedra, if a point is

in the positive half-space of all face planes, then it is interior to the object. If it is in the negative half-plane of any face plane, then it is exterior to the object. Otherwise, the point lies on the surface of the object (on one or possibly more faces). A surface point touches the object but is not considered to be interior to the object.

Thus, for polyhedral objects, an obel vertex point can be determined to be interior, exterior or on the surface using simple arithmetic operations without considering the location of the faces on the face planes or the actual intersections of face planes.

For convex objects defined by an analytic surface the location of the point relative to the object can often be easily determined by evaluating the defining expression for the surface. For example, an evaluation of $x^2 + y^2 + z^2 - R^2$ will yield 0 for a point on the surface of a sphere of radius R centered on the origin. A negative value indicates a point within the sphere and a positive value, exterior.

In 3-D, the most general second degree equation is:

$$Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J = 0 \quad (4-5)$$

Note that the terms have 3 formats, a constant times a coordinate, a constant times a coordinate squared and a constant times the product of a coordinate and a coordinate value from another dimension. Generating the terms of the expression for the vertices of a child obel using simple

Arithmetic on parent values allows evaluation for vertices requiring only simple arithmetic. This can be accomplished as illustrated in Figure 4.6. In 4.6(a) the value of Gx' required for evaluation of the child vertices is desired as a simple function of Gx from the parent. It is computed as follows:

$$Gx' = G(x+e) = Gx+Ge = Gx+G(2^{(m-n)}), \quad (4-6)$$

where m is the level of the lowest level in the universe where $e=1$ and n is the level of the child object. The first term is given and the second is a shift of the constant G . Thus, a shift by $m-n$ followed by an add will be sufficient.

In Figure 4.6(b) the value of Ax'^2 is desired from Ax and Ax^2 . The value of Ax' will be needed later and is calculated as above. The Ax'^2 value can be evaluated by:

$$\begin{aligned} Ax'^2 &= A(x+e)^2 = Ax^2 + 2eAx + Ae^2 \\ &= Ax^2 + (2^{(m-n+1)})Ax + A(2^{2(m-n)}) \end{aligned} \quad (4-7)$$

Two shift operations and two adds are required in addition to the shift and add to compute Ax' .

More sophisticated objects involving third order operations require Kx'^3 as shown in Figure 4.6(c). The Kx' and Kx'^2 values are maintained as above with Kx'^3 calculated as follows:

$$\begin{aligned} Kx'^3 &= K(x+e)^3 = Kx^3 + 2eKx^2 + e^2Kx + eKx^2 + 2e^2Kx + Ke^3 \\ &= Kx^3 + (2^{(m-n+1)})Kx^2 + (2^{2(m-n)})Kx + (2^{(m-n)})Kx^2 \end{aligned}$$

$$+(2^{(2(m-n)+1)})Kx+(2^{3(m-n)})K \quad (4-8)$$

Five additional shifts and adds are used for a total of 8 each for this term.

Two cross product terms are illustrated in Figure 4.6(d), $Dx'y$ and $Dx'y'$. The first is computed from the parent values of Dx , Dy and Dxy as follows:

$$Dx'y = D(x+e)y = Dxy+eDy = Dxy+(2^{(m-n)})Dy \quad (4-9)$$

A shift and an add are needed. The second is computed by:

$$\begin{aligned} Dx'y' &= D(x+e)(y+e) = Dxy+eDy+eDx+e^2D \\ &= Dxy+(2^{(m-n)})Dy+(2^{(m-n)})Dx+(2^{2(m-n)})D \end{aligned} \quad (4-10)$$

Three shifts and adds are needed.

Moving from points in space (vertices of the obels) to consideration of the space occupied by the obel itself, three simple tests can be performed to determine status. First, if all vertex points of the obel are on or in the negative half-space of the same face plane (at least one) there is no intersection. For analytic surfaces a plane tangent to the surface can be used to demonstrate non-intersection although selecting such a plane may be difficult.

Second, if all vertex points are on the surface of, or interior to the object, the obel is enclosed by the object. For polyhedra, all vertices must be on or in the positive half-space of all face planes.

Third, if at least one vertex is interior to the object and at least one vertex is exterior, the obel intersects the object but is not enclosed.

Unfortunately, not all situations are included in these tests. For example, if there is at least one vertex on the positive side of each face plane the test obel may or may not intersect the object as shown in Figure 4.7 for 2-D. Obel 1 does not intersect while obel 2 does. Also, a case with all vertex points exterior to the object is not necessarily disjoint as shown by obel 2.

The concept of direction space will now be introduced. This is a classification space in which vectors are assigned membership by virtue of their direction in Euclidean space. In 3-D there are 8 direction octants in direction space corresponding to the 8 combinations of the signs of the three axis displacement values for a vector. A displacement of 0 is of either sign. Every vector belongs to at least one direction octant. Zero displacement assigns a vector to multiple octants. Within a direction octant, all vectors can be thought of as pointing in the same general direction. They are within a 90 deg. range bounded by planes intersecting pairs of axes and form an angle of 90 deg. or less with each other.

The term critical vertex will be used to denote a single vertex which can be tested to determine some characteristic of the entire obel. Two critical vertex types are defined,

the negative test vertex and the positive test vertex. Because of the orthogonal orientation of obels, it is clear that for each vertex, the vectors from the vertex to all other points in the obel are all members of a single direction octant (some also reside in other octants).

Every face plane has an associated negative test vertex (in test obels being compared to it). It is the same vertex in every obel and is determined by the orientation of the plane. It is defined to be the vertex such that vectors from it to any other point in the obel are members of the same direction octant as the normal of the face plane.

For face planes parallel to an axis, the surface normal resides in two direction octants. Two negative test vertices are defined. For face planes perpendicular to an axis it resides in four direction octants and four negative test vertices can be identified.

The positive test vertex is defined in an identical manner using the negative of the surface normal (into the positive half-space).

Assertion 4.5: For a face plane, if an associated negative test vertex is on the plane or in its negative half-space, the obel is disjoint from the object.

Proof: Consider the normal to the face plane. Clearly, any vector from the surface or a point in the negative half-space to a point in the positive half-space forms an angle of greater than 90 deg. with the normal. Thus, a

vector from the negative test vertex to an interior point cannot be in the same direction octant as the normal. But the vector from the negative test vertex to any obel point is in the same direction octant. Therefore, no obel point can be in the positive half-space. The obel is entirely in the negative half-space. Q.E.D.

Similar reasoning shows the entire obel to be in the positive half-space if the positive test vertex is on the plane or in the positive half-space.

The above tests of obel status for convex objects are summarized in Table 4.1. The first is the elimination of obels exterior to the bounding box. The second determines no intersection if the obel is entirely on the negative side of a face plane or tangent plane. Test 3 finds enclosure if the obel is on the positive side of each face plane.

Unfortunately, not all situations can be resolved by these tests. The problem arises when the surface changes direction octant membership as noted above in Figure 4.7. Obels not resolved could be given a P value and subdivided with an arbitrary decision made at the lowest level but a non-zero false-P rate would result.

Before continuing, a restricted situation will next be considered in order to simplify the analysis. A restricted region is a convex region of the universe in which all surface normals are in the same direction octant. According to the following assertion, for non-polyhedral objects test 2

of Table 4.1 can be performed more easily if the test obel is in a restricted region. The test obel is external if the negative test vertex is simply external (a tangent plane does not need to be found).

Assertion 4.6: For a test obel entirely within a restricted region, if the negative test vertex is on the surface of a convex object or external to the object, the obel is disjoint from the object.

Proof: Generate a vector from the vertex to any point interior to the object within the region. Since the region is convex the vector lies entirely within the region with one point on the surface of the object. Place a plane tangent to the surface at this surface point. Since the surface point is in the region, its surface normal must be in the same direction octant as that used to determine the negative test vertex. Clearly the vertex is on the tangent plane or in its negative half-space. Since the tangent plane can be considered to be a face plane of the object (in the limit), from Assertion 4.5 the obel is disjoint from the object.

Q.E.D.

The following shows that in a restricted region, the obel intersects if the negative test vertex is not found to be in the negative half-space of any face plane in the region.

Assertion 4.7: For a convex polyhedral object and for a test obel entirely within a restricted region, if the

negative test vertex is in the positive half-space of all face planes in the region, the obel intersects the object.

Proof: First note that the vertex is not specified as being in the positive half-space of all face planes in the object, only those in the region.

Assume that the negative test vertex is on the surface of the object or external to it. Select a point in the region interior to the object. The line from the vertex to this point is entirely interior to the region because it is convex. Since the line intersects a face plane and continues into the positive half-space the vertex must be in the negative half-space of this particular face plane or on the face, clearly a contradiction. The critical vertex must be an interior point. Q.E.D.

For non-polyhedral objects, the obel obviously intersects if the negative test vertex (or any obel point) is not found to be exterior or on the surface of the object.

Applying the rules of Table 4.1, the 2-D situation is found to be a special case as illustrated in Figure 4.8(a). The false-P rate is zero.

Assertion 4.8: For a 2-D convex polygon, if the test obel intersects the bounding box and for all edge lines the associated negative test vertex is in the positive half-plane (i.e., not on negative side of any edge line), the test obel intersects the polygon.

Proof: Assume that the test obel does not intersect the

polygon. If the polygon is subtracted from the bounding box as in Figure 4.8(b), four disjoint polygons (possibly empty) called corner regions result. Since the test obel intersects the bounding box but not the polygon, it must intersect one of these four regions. It cannot intersect more than one because the obel is convex and therefore a line connecting a point in the intersection of both sections would be contained entirely within the obel. This line must cross the polygon, however, contradicting the assumption of no intersection.

One of the vertices must actually intersect the corner region. It is sufficient to provide an algorithm to find it. Select any point within the intersection area. Move in x toward of the polygon until the obel edge is hit. From here move along the edge in y toward the polygon. Before the polygon is hit, another edge of the obel will be hit. This point is the negative test vertex.

Note that a line between the two intersections of the bounding box and polygon which define the corner region forms a convex polygon, a triangle which encloses the corner region and the associated edges. Since the object is convex, the connecting segment is guaranteed to be on the surface of or interior to the object. This is noted in Figure 4.8(c). Also note that the edge normals for the enclosed edges are all members of the same direction quadrant. This triangular region thus forms a restricted region as defined above. The associated negative test vertex is the above vertex (at least

one of them) in the corner region.

With the negative test vertex in a corner region, the test obel cannot intersect the edges which form another corner region. This insures that the obel is entirely within a region of uniform direction quadrant edges even though the entire obel may not be entirely enclosed by the corner region.

Drawing on Assertion 4.7, if the negative test vertex is in the positive half-plane of all face edges in the region, the obel intersects the polygon. This contradicts the assumption that it does not intersect the polygon and proves the statement. Q.E.D.

This result is useful when creating 3-D objects by layering 2-D slices and is also used later by the display algorithm.

It would, of course, be desirable to somehow extend this zero false-P result to 3-D. Consider the three orthogonal projections of a polyhedral object and of the test obel along the principal axes as shown in Figure 4.9(a). As can be seen in Figure 4.9(b) the silhouette of the projection forms a prism which encloses the object. The two face planes of the prism are faces of the bounding box (extended in 4.9(b) for clarity). All face planes are parallel to an axis.

Since the object is convex, the 2-D projection is convex. If this were not so, two points could be found within the object such that the line connecting them would

not be entirely within the object. The prism is therefore convex. Since it entirely encloses the object the three such prisms will be called the bounding prisms.

Clearly, if for any of the axes, the projection of the object (and therefore the bounding prism) is disjoint from the projection of the obel, the object and the obel are disjoint. For each dimension, the projection intersection case can be determined by applying the above 2-D techniques.

Obviously, this test will eliminate some false-P cases which would have a reduced value of E but will it eliminate all? The answer is that it will.

Assertion 4.9: For a 3-D convex polyhedral object, if (1) the test obel intersects the object's bounding box and (2) the three orthogonal axial projections of the obel intersect the corresponding projection of the object and (3) for all object face planes the associated negative test vertex is in the positive half-space, then the test obel intersects the object.

Proof: Consider what will be called the bounding object. It is the intersection of the three bounding prisms and is a polyhedron built with face planes from the bounding prisms. Note that all face planes of the bounding object will be parallel to an axis. Five bounding entities have now been defined, the bounding box, three bounding prisms and the bounding object.

The proof will consist of two parts, first showing the

test obel intersecting the bounding object and then the object itself.

Assume that the test obel does not intersect the bounding object. Since the projections of the obel and object intersect, the projections of the obel and the bounding prisms also intersect. Part of the bounding object must therefore lie within the square projection of the test obel for all three axes. Since it is assumed there is no actual intersection, in each dimension the part of the bounding object the projection of which intersects the obel projection must lie in front of or behind the test obel. There are eight such cases, corresponding to the positive or negative direction for each of 3 axes. It will be sufficient to analyze one case; the others are easily disposed of in similar fashion.

Consider the case shown in Figure 4.10(a). The object is in the positive direction relative to the test obel in all 3 axes. Part of the bounding object must exist in each of the three volumes numbered 1 through 3. The numbered faces are formed by face planes of the object bounding box.

Consider a translated set of axes centered on the forward most vertex (vertex 7) of the test obel as shown in 4.10(b). Select a point in each of the three volumes that is also within the bounding object and label them points 1 through 3 as noted.

Since all three points are within the bounding object,

they must be in the positive half-space of all bounding object face planes. Since the origin of the translated set of axes is not interior to the bounding object, it must be on or in the negative half-space of a face plane (at least one). Now consider this face plane. It must be parallel to one of the axes. Assume that it is parallel to the z axis. Its projection on the x - y axis is shown in Figure 4.10(c). Note that points 1, 2 and 3 are restricted to quadrants IV, II and III, respectively.

Since points 1 and 2 must be in the positive half-space, the projection must have a negative slope in the x - y plane. Because of this and because the origin cannot be in the positive half-space, the projection cannot intersect quadrant III and the face plane cannot intersect volume 3. Thus point 3 cannot be on the positive side of the plane and such a point cannot be in both volume 3 and the bounding object. Similar arguments can be made for face planes parallel to the x and y axes and for the other 7 arrangements of points 1 through 3 relative to the obel. This contradicts the assumption and proves that the obel and bounding object intersect.

The second part of the proof begins by noting that the object itself separates a bounding prism into two (possibly empty) parts. See Figure 4.11. Since the edges of the projection correspond to actual object edges, the edges can be connected with lines entirely within the object (it's

convex) to form a surface (not necessarily planar and possibly not unique) which completely separates the bounding prism. If the object is removed from the bounding prism, two disjoint subsets of the bounding prism remain. Clearly, no line can intersect both subsets without crossing the surface and therefore intersecting the object.

Next consider the object surfaces that touch these subsets. Figure 4.12 shows an object edge between two object face planes from different direction octants. A plane parallel to the y axis is placed so as to intersect the edge. Since it is tangent to a convex object, no part of the object can exist in its negative half-space. Thus, its projection on the x - z plane forms an edge of the projected polygon and the plane is a face plane for the bounding prism parallel to the y axis.

The change of sign of the y component of the surface normals at an edge will always give rise to an edge in the x - z plane. A vertical object face plane has a y component of 0 and will always form a projected edge. Two transitions will form the same projected edge. Also note that if multiple transitions occur across an edge, it will show up in multiple projections.

Since the y transition edges will form the edges of separation between the two bounding prism sections (with the object removed), the object face planes in the upper section (plus y in this case) must have positive y normal components.

The lower section must have negative normal components. Face planes with a zero y normal component are a face of the object and a face of the bounding prism and are subtracted out. They do not touch the remaining sections across a face.

Applying this in the two remaining axial directions, if the object is subtracted from the bounding object, 8 sections (possibly empty) remain. Within any one section, the faces which correspond to an object face will all be of the same direction octant. Also, because a separating surface exists between every pair of corner regions, no line can intersect two corner sections without intersecting the object.

The remainder of the proof parallels that for Assertion 4.8. The test obel is known to intersect the bounding object. Assume that it does not intersect the object itself. It must intersect but a single section of the bounding object. If it touched more than one, a line entirely within the obel could be drawn to connect the two sections, crossing through the object and violating the non-intersection assumption.

Next, the negative test vertex for the direction octant for the touching object faces must be within the region. The algorithm from the proof of Assertion 4.8 demonstrates this.

As above, since the negative test vertex for the section is in the positive half-space of all face planes in the region (a given) drawing on Assertion 4.7 the obel must intersect the object. This contradicts the assumption that

is not intersect the object and proves the statement.

number of point-to-line distance computations must be made to incorporate this test. For an E node, the minimum added to E is thus zero. Otherwise some number of edge intersection lines must be compared to their associated 2-D test vertices. Required for F and P nodes and worst for E nodes would be a test for each axis of each edge intersecting two face planes from different direction octants to span the plus and minus signs for the normal vector components along the axis. This is the total number of edges which appear in the 3 axial object projections. It is possible for one edge to appear in all three projections.

that edges formed between 2 faces of the same direction octant are not considered. A worst case upper bound of 3 is the number of edges can be set by assuming all edges in all direction quadrants and all appear three times.

The total number of edges in a polyhedra can be related to the number of faces and vertices by examining Euler's Law for polytopes:

$$n_0 - n_1 + n_2 + \dots + (-1)^{N-1} n_{N-1} = 1 - (-1)^N \quad (4-11)$$

where n_i is the number of entities of dimension i (n_0 = vertices, n_1 = edges, n_2 = faces, n_3 = solids, etc.) and N is the number of dimensions.

In 2-D (4-11) requires the number of vertices to equal the number of edges. In 3-D, the following relationship holds: $\text{edges} = \text{faces} + \text{vertices} - 2$

An upper bound for point-to-line calculations is thus:

$$3(\text{faces}) + 3(\text{vertices}) - 6$$

After the projection tests have been performed, the vertex-to-plane tests may need to be performed. As with the 2-D case, all faces, rather than just those in a known direction octant may require checking.

This technique cannot be directly applied to non-polyhedral objects, however, because the negative vertex test is not guaranteed to be correct unless the test vertex is in a restricted region. This is not a problem for polyhedra because the negative half-space of a face plane is a restricted region.

One solution is to require that the universe containing the object be able to be divided into 8 restricted regions. Test obels will be required to be enclosed in a single region. This would also facilitate polyhedral object generation because only face planes in the region need be tested.

A problem is the false-P rate. Using restricted regions, the final node value will be correctly determined from the tests but since a test obel must lie entirely within a region, obels may be required to be subdivided along planes separating the regions. An option in some situations is to

further restrict objects to what will be called standard form. The 8 regions are the 8 octants of the universe. Except for the trivial case of a single node (F or E) octree, no obel will be declared P and subdivided because it spans multiple direction octants.

Many objects of general interest can be created by transforming objects into desired instances using geometric operations after they have been generated in standard form.

For a standard form polyhedron, the procedure is as follows:

```
if CASE='DISJOINT' in any dimension then STATUS<-'E' else
if DISTANCE(NEG_TEST_VERTEX, PLANE)<=0 for any plane PLANE
  in direction octant then STATUS <- 'E' else
if DISTANCE(POS_TEST_VERTEX, PLANE)<0 for any plane PLANE
  in direction octant then STATUS<-'P' else STATUS<-'F'
```

The function DISTANCE(VERTEX, PLANE) returns the signed perpendicular distance from vertex VERTEX to plane PLANE.

The first decision is based on the bounding box test requiring a minimum of one comparison or a maximum of 2N if true and always 2N if false. In the following analysis, it is assumed that no face planes are perpendicular to an axis. Allowing them actually speeds up the algorithm because they form part of the bounding box but complicates the analysis.

Empty nodes in an octant for which there are no associated planes will be eliminated by the bounding box. Thus, empty nodes which intersect the bounding box will require a distance to plane computation to be performed at

least once and possibly as many times as there are faces in that direction octant (certainly less than the total number of faces). An E node will therefore require a minimum of one comparison with an upper bound of $2N$ magnitude comparisons plus one vertex-to-plane distance computation for each face in the octant.

For P or F nodes, all planes in the octant must have been checked against the negative test vertex. A P node will require a minimum of 1 additional check and a maximum of the number of faces in the octant. An F node will always add a distance computation for each face in the octant. The requirements are summarized as follows:

	----- MINIMUM -----		----- MAXIMUM -----	
	mag. compare	vertex-to- plane	mag. compare	vertex-to- plane
E	1	0	$2N$	faces
P	$2N$	faces+1	$2N$	$2(\text{faces})$
F	$2N$	$2(\text{faces})$	$2N$	$2(\text{faces})$

A lower bound for 3-D can be found by assuming that all leaf nodes have a status of E. A lower bound of $13C/8 - 5/8$ magnitude comparisons and $C(\text{faces})/8 - \text{faces}/8 + C/8 - 1/8$ vertex-to-plane distance calculations performed where the value of "faces" can be taken to be the minimum number of faces in an octant. All F leaf nodes can be assumed in order to find an upper bound. This gives $6C$ magnitude comparisons and $2C(\text{faces})$ vertex-to-plane distance calculations where

faces can be taken to be the maximum number of faces in an octant or the total number of faces (if a breakdown by direction octant is not available) for a worse but still valid upper bound.

For convex analytic objects in standard form, the following procedure will generate no false-Ps:

```

if CASE = 'DISJOINT' in any dimension
then STATUS<-'E' else
if SURFACE_TEST(NEG_TEST_VERTEX)='EXTERIOR'
  for all direction octants
then STATUS<-'E' else
if SURFACE_TEST(POS_TEST_VERTEX)='INTERIOR'
  for all direction octants
then STATUS<-'F' else STATUS<-'P'

```

The first decision is, again, the bounding box test. The second is a check ("SURFACE_TEST") to see if the negative test vertex as specified by the octant of the obel is outside the object (including the surface). An E node requires a minimum of 1 magnitude comparison and a maximum of 2N magnitude comparisons plus one surface test. A P or F node requires this maximum plus a second surface test.

Taking all leaf nodes to be E, a lower bound of $13C/8$ magnitude comparisons plus $C/4$ surface tests is required. If all leaves are taken to be F, an upper bound of 6C magnitude comparisons and 2C surface tests will be required.

4.2.3 Object Generation Examples

Example 4.1: Generate the octree for the half-space defined by $Ax + By + Cz + D = 0$.

Solution: The positive (interior) half-space will be the side containing points which are at a positive distance from the plane when the defining equation is evaluated.

First the root node is created and its address pushed on a node address stack. The distances for the vertices of the universe are generated from (4-3), placed in vector DIST and passed to procedure HALF_SPACE (see below).

The vertex numbers of the critical vertices have been predetermined and are contained in constants POS_TEST_VERTEX and NEG_TEST_VERTEX.

As the procedure continues, if the positive test vertex is in the positive half-space, the obel is enclosed and the associated node is marked F by procedure NEW_NODE. If not, the negative test vertex is checked to detect an E node. If neither is the case, a vertex exists on either side of the boundary, resulting in a P node. Procedure NEW_NODE creates the children and attaches pointers to the node. The addresses of the children are pushed on the node address stack (in reverse sequence order) for later use. HALF_SPACE is then called to evaluate the children. Procedure CHILD_DIST calculates the vertex-to-plane distance vector for a child based on the parent values, using (4-4).

The procedures generalized to N-dimensions are outlined as follows. Maintenance of a tree depth value and forced node termination at the lowest desired level have been left out for brevity.

```
procedure HALF_SPACE(DIST)
```

```
  vector_2N DIST
```

```
  if DIST(POS_TEST_VERTEX) >= 0 then NEW_NODE('F') else
```

```
  if DIST(NEG_TEST_VERTEX) <= 0 then NEW_NODE('E') else
```

```
    begin
```

```
      NEW_NODE('P')
```

```
      for CHILD <- 0 step 1 until 2N-1 do
```

```
        HALF_SPACE(CHILD_DIST(DIST,CHILD))
```

```
    end
```

```
procedure NEW_NODE(STATUS)
```

```
  /* Pop node address and attach STATUS. Attach 2N
     unmarked child nodes and push addresses on
     stack in reverse order if 'P'. */
```

```
procedure CHILD_DIST(DIST,CHILD)
```

```
  /* Returns distance vector for child CHILD of
     parent distance vector DIST. */
```

Example 4.2: Generate the octree for a sphere of radius R centered on the center of the universe.

Solution: The sphere contains all of the points at a distance less than R from the center of the universe defined by:

$$d^2 = (x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2 \quad (4-12)$$

Determining the value of d for obel vertices is not easily performed with simple arithmetic but the individual terms of (4-12) and therefore d^2 are easily computed using (4-7) with $A=1$. Since squared positive numbers are monotonically increasing, and both d and R are positive, $d^2 < R^2$ if $d < R$. Thus d^2 can be compared to the radius squared to determine if a particular vertex is interior or exterior to the sphere.

A procedure similar to HALF_SPACE called SPHERE is presented below. The distance vector now holds the distance values squared and is called DIST_SQ. The radius squared is in constant RAD_SQ. Procedure CHILD_DIST_SQ computes the terms of (4-12) based on (4-7) and sums them to compute the distance squared values for the child vertices.

The vertex distances are all initialized to $3e^2/4$ where e is the length of an edge of the universe. A few items have been neglected in SPHERE. First, the 8 octants of the universe correspond to the 8 direction octants and contain different critical vertices. New values for POS_TEST_VERTEX

and NEG_TEST_VERTEX are needed at the 7 times during algorithm operation when the level 1 node is changed. Also, a mechanism is needed to force the root to be a branch node since it (and it alone) spans direction octants.

The DIST_SQ vector must be expanded to maintain the component and component squared values. Bottom level node termination has again been neglected.

Procedure SPHERE is as follows:

```

procedure SPHERE(DIST_SQ)
vector_2N DIST_SQ
if DIST_SQ(POS_TEST_VERTEX) <= RAD_SQ then NEW_NODE('F') else
if DIST_SQ(NEG_TEST_VERTEX) >= RAD_SQ then NEW_NODE('E') else
begin
NEW_NODE('P')
for CHILD <- 0 step 1 until 2N-1 do
SPHERE(CHILD_DIST_SQ(DIST, CHILD))
end

```

Example 4.3 Generate the octree for a cylinder of radius R centered along a line through the center of the universe parallel to an axis.

Solution. Procedure SPHERE can be employed if CHILD_DIST_SQ is simply modified so that the term of (4-12) in the direction of the centerline is eliminated. The vertices of the universe are initialized to a value of $e^2/2$.

4.2.4 Concave Objects

Most of the intersection tests developed above are invalid when dealing with concave objects. About the only rules which can be employed are the bounding box test and a determination of intersection if a vertex point is internal and another is external. The given method of interior/exterior determination is invalid for concave polyhedra, however.

Given the current state of development, the strategy for concave objects should probably begin by attempting an indirect generation. Some objects can be decomposed into multiple convex objects. After generating the parts, they are unioned together. The set operations of intersection, difference and negation could also be employed. Many interesting objects can also be created by sweeping a convex object.

If a concave object is reasonably well behaved, some general strategies can be developed. The following are only guidelines in that bizarre objects can often be envisioned which would cause problems. Included are zero thickness places or surface wraparound creating unknown interior holes, etc.

A few additional testing capabilities must be added. It is assumed that an interior point (any point) within the object can be easily found. A method must be provided to

determine if a point is interior or exterior. This is not generally as simple as for convex objects. A popular technique is to extend a line from the point to a known exterior point. If the line cuts the surface an odd number of times, the point is interior. Some way must also be found to determine if the surface of the object cuts the surface of the obel. This is often simplified because obel surfaces are planes perpendicular to an axis.

Given this and if the object is bounded, encloses volume, has no interior holes and no disjoint parts, the following procedure can be used:

```
if surface of object cuts test obel then STATUS <-'P' else
if any obel vertex interior to object then STATUS<-'F' else
if any object point interior to obel then STATUS<-'P'
else STATUS<-'E'
```

In the first test, if the object cuts across the obel surface, they intersect but the obel is not enclosed. This assumes that the surface is legitimate, separating the interior and exterior. For the second test, since the surfaces don't cut each other, obel enclosure is tested. We can pick any obel point, a vertex being convenient. If inside the object, it should be completely enclosed since no interior holes are allowed. For the third test a check is made for the opposite condition, the object enclosed by the test obel. Any point in the object is tested for enclosure by the obel. If so, it is a partially enclosed node. If not, the object and obel are disjoint.

If interior holes and disjoint parts are allowed, a method of determining if any part of the surface is within the test obel is needed. The following general procedure can be followed:

```
if any surface of object interior to test obel
then STATUS<-'P' else
if any obel vertex interior to object
then STATUS<-'F' else STATUS<-'E'
```

The first test checks if any part of the object surface is within the obel. If yes, there is partial intersection. If not, the obel is either completely enclosed or completely disjoint. Any obel point such as a vertex is used to decide.

For both of the above concave object procedures a zero false-P rate can be expected for well behaved objects but some tests may be difficult to perform.

4.3 Object Properties

One of the strong advantages of Octree Encoding relative to other SMSs is the efficiency of object property measurement. This is a consequence of spatial pre-sorting. In an actual system the real-time generation of mass properties should not be difficult to achieve.

Of the following, volume, surface area and separation of disjoint parts are 3-D extensions of published quadtree techniques.

4.3.1 Volume

The volume of an octree object is easily calculated by simply counting the number of F leaf nodes at each level, multiplying by the volume of an obel at that level and summing. If an error tolerance is allowed, the levels can be examined from the root until the expected value of the volume is within limits. A minimum and maximum volume is calculated. Only F nodes are summed into the minimum while P nodes are also summed into the maximum.

4.3.2 Surface Area

The surface area of an octree is determined by checking the face neighbors of each F node. If the neighbor is E, the area of the face is summed into the surface area. If a neighbor is P, the F node is subdivided into 8 children with a status of F which are then processed.

This method computes the surface of the octree object, not the original input (real) object (or input objects used to create it). If a more precise measurement is needed, analytic surfaces could be fitted to the octree object or perhaps the application data base could be examined.

4.3.3 Center of Mass

If an object with uniform density is divided into n regions, no two of which overlap, having volumes $V_1, V_2, \dots V_n$ and centers of mass $(x_{1,1}, x_{2,1}, \dots x_{N,1}), \dots (x_{1,n}, \dots x_{N,n})$, the object center of mass, $(x_1, x_2, \dots x_N)$ can be calculated as follows:

$$x_i = \frac{\sum_{j=1}^n (V_j x_{i,j})}{\sum_{j=1}^n V_j} \quad (4-13)$$

Clearly, F obels are the needed disjoint regions for an octree object with the obel centers being the center of mass. The centers can be generated using only simple arithmetic as shown above. Because the edge size of an obel is a power of 2, the volume is a power of 2. The multiplication by obel volume can therefore be performed by shift operations.

In operation, the center of mass algorithm maintains two variables for each dimension, corresponding to the numerator and denominator of (4-13). For each F node, the center is multiplied (via shifts) and summed into the numerator. The volume is summed into the denominator. After all nodes have been processed, the variables are divided to obtain the center of mass components.

As with the volume calculation, a maximum error can be calculated and compared to a user supplied value so computation terminates when within tolerance.

4.4 Moment of Inertia

The moment of inertia, I , is defined as follows:

$$I = \sum_{i=1}^n (M_i R_i^2) \quad (4-14)$$

where M_i is the mass of a particle of mass i and R_i is the perpendicular distance from the particle to the axis of rotation.

For a homogeneous octree, the mass of an F node is proportional to the volume and is, again, a power of 2, allowing the multiplication to be performed by shifts. The distance squared value is easily computed using simple arithmetic, as noted above, if the axis of rotation is (or has been made through rotation) parallel to a coordinate system axis.

A problem arises, however, because of the nonlinearity of the distance squared. The above formula assumes point masses. Errors result if the center of a distributed mass is used. This is minimized by subdividing the F nodes down to lower levels before performing the summation.

An approximation to the moment of inertia is thus easily computed with simple arithmetic by summing the product over F nodes or their (forced) implied children.

4.3.5 Segmentation of Disjoint Parts

It is sometimes desirable to separate an octree containing multiple disjoint parts into multiple octrees, one for each part. The algorithms which have been developed to perform segmentation check each face between F obels. A table is kept of object numbers. If an F node is discovered which has not already been given an object number, the face neighbors are checked for object numbers. If none are numbered, it and the full neighbors are given a new number that's also entered into the table. If, among the neighbors, there is a single object number, it is given to the obel and any unnumbered neighbors. If the node and its neighbors have been given conflicting numbers, equivalence pairs are generated. When all F nodes have been processed, a second phase resolves the equivalence pairs and a third phase attaches the final labels. The nodes can be separated into multiple octrees by phase three.

4.3.6 Interior Voids

To obtain the number of interior voids in an object, it is negated and then segmented into its disjoint parts. The number is generally the number of parts minus one. The part external to the object can usually be determined because it touches a vertex of the universe. It could be deleted,

leaving just the negated interior voids as solid objects.

A space filling operation is performed if one or more of the negated interior sections is unioned back into the original object.

4.3.7 Correlation

A measure of the correlation between two objects can be determined by calculating the fraction of a volume containing the objects which has the same status (E or F) in both objects. The measuring volume would typically be the bounding box of the union of the two objects. This measure of correlation would then be:

$$(\text{volume}((\bar{A} \cap \bar{B}) \cap \text{BOX}) + \text{volume}(A \cap B)) / \text{volume}(\text{BOX}) \quad (4-15)$$

where A and B are the objects and BOX is the bounding box.

4.4 Boolean Operations

The Boolean operations are the "regularized" operators [11]. An intersection must contain volume, for example. Thus the intersection of two 3-D objects that touch along a surface is the null set. Conveniently, this is the normal result of the tree manipulation operations presented here.

Algorithms that perform the Boolean operations of union,

intersection and difference have been developed for quadtrees [12] and are directly extendable to 3-D octrees. They are not repeated here for brevity. It is simply noted that simultaneous traversal of multiple trees visits the same perfectly aligned sections of space in each object. The status values can be examined during traversal to generate the output status values.

The negation operation can be performed by simply changing all F nodes to E and vice versa. If performed often, algorithms could optionally negate the interpretation of leaf values. No node values would need be changed.

4.5 Overlays

The sections which follow will outline more advanced techniques for performing transformations and operations such as translation, scaling, rotation, swept volume generation, interference detection and display. If there is one tool that forms a spine through most of these algorithms, that tool would be the overlay. From an implementation viewpoint its importance would be difficult to overstate.

An overlay is a set of contiguous nodes at a single level, drawn from an input tree, that is guaranteed to spatially enclose an associated obel, the target obel, from another tree, usually an output tree being generated. By examining the overlay, a decision (perhaps not a final

decision) can be made as to the status of the associated node.

The overlay can be thought of as a template containing a number of obels which cover a section of space. Within any overlay there is a single obel in a fixed location relative to the overlay such that if the origin of the target obel is within it, the target obel is covered by the overlay. The shape of the overlay, the number of obels it contains, the level of its obels relative to the level of the target obel, etc. depend on the specific algorithm. By maintaining a set of active neighbors, the overhead of pointer chasing through the tree in order to examine neighbors is reduced.

From a computational growth viewpoint the savings are not significant because the expected value of the average traversal length required to locate a neighbor is a constant [13]. From an implementation viewpoint, however, the savings can be substantial because far fewer memory references are needed. Also, the maximum length of time which need be allowed for a processor to generate and evaluated a sub-overlay is greatly reduced.

Operating under the self-imposed restriction of only simple arithmetic the essential idea behind most of the following algorithms is to set up an imaginary output tree and then proceed to examine its obels for spatial interference with the input tree or trees. By examining intersecting obels a status value can be given to the output

node. The most basic primitive operation that must be performed is interference detection between the output node and the nodes in the input tree. If a high level of overall performance is to be expected, obviously this must be performed very efficiently.

Conventional CSG systems perform an interference test between an object and a single primitive in a second object by mathematically comparing the primitive to each primitive in the first object. Depending on the results needed, processing may stop if an intersection is found. If the exact nature of the interference is required or if no intersection is found, the number of comparisons is equal to the number of primitives in the first object. For a complete determination of intersection between the two objects, the number of primitive pair checks can equal the product of the primitive counts in the two objects. For similar objects, interference detection is quadratic in object complexity.

Because the number of cubes in an octree can be expected to be much larger than the number of primitives in an approximately equivalent object in CSG format, quadratic growth would seem to insure extremely poor relative performance. It is at this point that the admittedly large effort and expense of using and maintaining a spatially pre-sorted data structure comes to fruition. It is not necessary to check all nodes in the input tree when the status of an output tree node is needed. Only those which

can possibly intersect that node, i.e., those in its overlay, are tested. In the following algorithms, these nodes are fixed in number and easily determined. Thus, the number of tests is limited to the product of a fixed, usually small, number and the value of C for the output tree. The overlay is simply a convenient and efficient logical structure in which to keep track of them within an algorithm (or within a processor).

Advantage cannot be taken of this technique in most current SMS because the primitives are not and usually cannot be spatially pre-sorted.

In 3-D a set of 8 obels all of which touch at a point will always completely enclose a randomly located target obel of the same size or smaller if orthogonally oriented (not rotated). The overlay associated with the target obel thus contains information concerning 8 obels (the overlay obels). The overlay data for each item may contain a pointer to a node in the input tree or could contain the status value of the terminal node from a higher level which contains the associated space.

Two tools are needed to evaluate the status of the target obel. First, the overlay obels which actually intersect must be determined and second, a status decision must be drawn from the status values of the intersecting ones. The first depends on the specific algorithm and can become quite involved. The second is simple. If all

intersecting nodes have the value F, the node for the target obel can safely be given a status value of F. The same is true for the status value E. Otherwise, the target obel is subdivided and each of its children processed in like manner. At the lowest desired level a terminal value is forced.

Another tool is needed to generate a sub-overlay for a specific child of the target obel. Since the overlay encloses the parent, it and some subset of its children (the children of its overlay obels) cover the child. Also, by definition, the overlay covers the target obel regardless of its location or orientation (as allowed by the algorithm) and therefore a sub-overlay containing the same number of obels as the overlay is guaranteed to cover the child target. Thus, the sub-overlay is identical in structure to the overlay with the new overlay obels being drawn from the set of children of the overlay obels.

Sub-overlay generation begins by determining the origin of the new target. Since it must be within a specific overlay obel, the exact enclosing child of the overlay is determined. The selection of the remaining sub-overlay obels is fixed by the structure.

The entire operation is easily performed using simple arithmetic. In practice, it is usually a table driven operation indexed by the relative distance between the origin of the overlay and the origin of the new target.

Information concerning the output node and the overlay

are passed from the parent to the children through a queue for breadth-first traversal or a stack for depth-first traversal.

4.6 Geometric Operations

The geometric operations are among the most basic SMS functions needed. For translation, scaling and rotation, an informal description of the algorithm will be described. The formal procedures are presented in [5].

4.6.1 Translation

The goal of the translation algorithm is to convert a tree representing an N-dimensional object and a movement vector into a new tree representing the translated object. The movement vector specifies a translation value to some precision, i.e. at some level for each dimension. An informal description of the algorithm follows.

The process begins by generating an augmented overlay universe composed of the "old" universe containing the original object and a number of empty universes. It is put together so the "new" universe containing the translated object is covered by the old universe. The translation vector specifies the alignment of the new universe relative to the old.

Beginning with the root of the new universe, the basic strategy is to traverse the implied tree for the new object and generate node values as outlined above while traversing the tree representing the old universe. If a terminal value for a node in the new tree is generated, no descendants of that node need be considered. If an ambiguity exists and, therefore, the status cannot be resolved, a P node is generated and its children are generated using the same procedure. The overlay obels present a picture of the section of the old universe that covers the new obel. They can be at any level relative to the target obel. The lower they are below the target, the more numerous but the more accurate the result (lower false-P rate). The output tree will be more nearly the final reduced tree. This comes at an increase in memory use and computation. The translation algorithm normally uses an overlay with 2^N obels at the same level as the corresponding obel in the new universe.

Figure 4.13(a) illustrates the overlay in one dimension. Distance is positive to the right. Three obels are shown. The target obel has an edge distance of e . Its lower end is the local origin. The overlay is made up of two adjacent obels of the same size from the old universe connected at the overlay center. The offset value is the distance from the local origin to the overlay center. The value of offset is limited to $0 \leq \text{offset} < e$.

The orthogonal block techniques from the object

generation sections above are used to determine intersection. The target obel corresponds to the convex object and the overlay obels the test obels. One difference is that the target obel does not remain fixed for all time. It subdivides. This presents no problem because the point-to-edge or point-to-plane distance values are easily updated for target subdivision using the techniques for test obel subdivision. The edges and faces of target obel children are parallel to the parent edges and faces.

Figure 4.13(b) shows the configuration of a 2-dimensional overlay. The 8 overlay cubes and the target cube for 3-D translation are illustrated in Figure 4.14.

As generated, node by node, the output tree will be correct but will not, in general, be identical to the reduced tree. False-P nodes will be generated when a target node later determined to be F or E intersects a target obel of status P. If a reduced tree is needed, a second pass reduction phase is required.

4.6.2 Scaling

Scaling an object by a power of two in all dimensions is accomplished by adding or deleting levels at the root. An object is halved in each dimension by adding one level at the top. The new root points to one branch node, the old root, and $2^N - 1$ empty terminal nodes. The scaled down universe can

be located in any of the first-level obels in the new universe.

In like manner, selecting one of the first level nodes to be a new root doubles the size of everything within it. To double the size of an arbitrary section of space, it is translated into a first level obel which is then expanded to fill the new universe.

Objects can be expanded or reduced by any power of two by, in effect, repeated expansion or reduction by a factor of 2. These scaling operations can be accomplished by manipulating a very small number of nodes at the top of the tree. The vast bulk of the data values are left unchanged.

Scaling by a factor other than a power of 2 is accomplished using an overlay scheme similar to that used for translation. The target obel, however, may be smaller than the overlay in one or more dimensions. In addition, a single set of offset values cannot be used. The offset vector must be computed independently for each child of the target obel.

4.6.3 Rotation

Rotation by 90 degrees about an axis can be performed by a simple reordering of the nodes in the tree. For an octree, if the center of rotation is the center of the universe, rotation by 90, 180 or 270 degrees or reflection across a plane through the center parallel to a face of the universe

or oriented at 45 degrees is accomplished by reordering or, within an algorithm, a change in the traversal sequence.

For rotation about an arbitrary point, the object is translated to the center of the universe, rotated, and translated back.

Rotation by an arbitrary angle is somewhat more difficult. In 2-D, the overlay arrangement for a 0 to 90 degree rotation is shown in Figure 4.15. Nine overlay obels are required. The local origin of the target obel is always in overlay obel B. Clearly, for 0 deg. to 90 deg. rotation, the target will be covered by the overlay in dimension 1. In dimension 2, the maximum reach of the target is $1 + \text{SQRT}(2)$ which is less than 3, the height of the overlay. Coverage is thus guaranteed.

Again, the techniques from the object generation sections are used to determine overlay obel intersection.

For an arbitrary 3-D object rotation, three operations could be performed using a slightly modified 2-D algorithm, one for each axis. An even simpler method is to perform multiple skew operations. This approach may be generally undesirable, however, because at the lowest levels aliasing products are generated on the object surface when nodes are forced to be terminal. The object is essentially being redigitized from a previously digitized object. This tends to corrupt the surface and is compounded by the repeated operations that are required. This is minimized by computing

nodes to lower levels, by performing operations in a single pass and by regenerating instances of an object from the original octree model each time it is moved or changed (rather than incremental movement of a single object).

Because of this, 3-D rotation in one pass is preferred. This is accomplished by extending the 2-D scheme into 3-D by the use of a 4 by 4 by 4 overlay.

The necessity for 4 rather than 3 overlay obels per dimension is illustrated in Figure 4.16. The maximum distance between two points in the target obel is the length of a diagonal through the cube or $\text{SQRT}(3)$ (about 1.7) assuming an edge size of 1. The origin is restricted within a range of 1 unit in each dimension. The overlay must span 4 units, however, because the origin is not guaranteed to be at an end of the projection width. If located within, a distance on either side of the centering obel must be allowed. In the diagram, distance A to the left of the center projection must be allowed and B to the right. In the case shown, the value of A is less than B and therefore less than 1. A single unit to the left is sufficient. The value of B could, however, be greater than 1, requiring two units to the right. For the case shown, the origin is located in the overlay obel labeled F. Depending on the location of the origin within the projection, the origin could be restricted to any of the center four, F, G, J or K.

5.4 Concatenated Geometric Operations

Using homogeneous coordinates [14] any number of sequential linear operations ($x' = A_1x + B_1y + C_1$, $y' = A_2x + B_2y + C_2$) can be concatenated and reduced to a single matrix coefficients (3 by 3 for 2-D or 4 by 4 for 3-D). This, of course, includes the geometric operations of translation, scaling and rotation. The composite transformation can then be performed in a single matrix multiplication. This can be expressed as follows for 2-D:

$$[x', y', 1] = [x, y, 1] \begin{bmatrix} A_1 & A_2 & 0 \\ B_1 & B_2 & 0 \\ C_1 & C_2 & 1 \end{bmatrix} \quad (4-16)$$

Reversing for the moment the direction of data flow in an overlay operation (the target obel will now "generate" the overlay) the above concatenated transformation will be performed if a parallelogram shaped target obel is used as shown in Figure 4.17. The target universe is assumed to be of unit edge length. The matrix coefficients determine its location, orientation and shape.

The C coefficients specify the location of the origin of the universe. The A coefficients determine the length and slope of the lower and upper edges relative to the overlay while the B coefficients determine the left and right edges. The original universe containing point (x,y) is shown in Figure 4.18(a). In 4.18(b), the transformed x value is shown. The C_1 value gives the origin offset. The

AD-A117 450

RENSSELAER POLYTECHNIC INST TROY NY IMAGE PROCESSING LAB F/G 9/2
OCTREE GENERATION, ANALYSIS AND MANIPULATION.(U)

UNCLASSIFIED

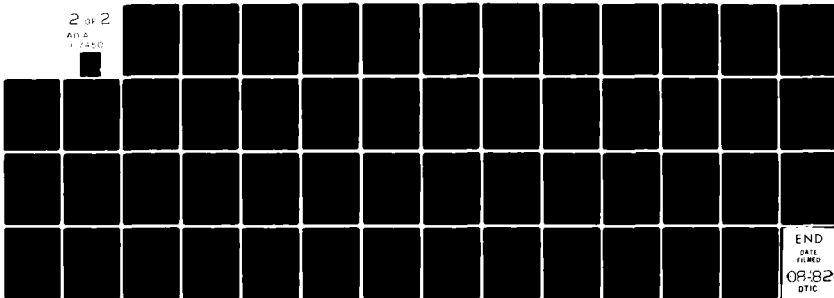
APR 82 D MEAGHER
IPL-TR-027

N00014-82-K-0301

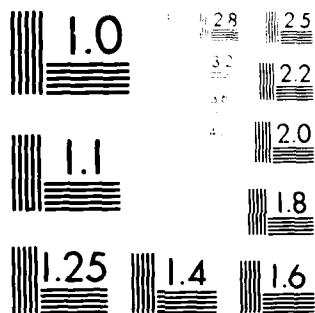
NL

2 OF 2

AD-A
17450



END
DATE
FILMED
08-82
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

specifies the fraction of the lower edge of the target universe spanned between the origin and the projection of the point on that edge. Since the projection of the edge on the x' axis is A_1 , this displacement, when projected on the x' axis is A_1x .

The value of y is similarly the projection on the left edge which projects a distance of B_1 on the x' axis. The x' component due to the value of y is thus shown to be B_1y . The value is the sum of the three components or $x' = A_1x + B_1y + C_1$. The value of y' is computed in like manner as $y' = A_2x + B_2y + C_2$.

For a universe with a non-unit edge, the matrix coefficients are simply multiplied by the edge value to generate the target universe parameters.

The matrix product for concatenated linear transformations in 3-D can be expressed as follows:

$$[x', y', z', 1] = [x, y, z, 1] \begin{bmatrix} A_1 & A_2 & A_3 & 0 \\ B_1 & B_2 & B_3 & 0 \\ C_1 & C_2 & C_3 & 0 \\ D_1 & D_2 & D_3 & 1 \end{bmatrix} \quad (4-17)$$

The geometric values for the target obel used to perform equivalent transformations are shown in Figure 4.19. An additional dimension is added to account for the new terms and the third equation.

Returning to the original data flow from overlay to target obel, the new obel is generated from the overlay when it is deformed according to the inverse of the coefficient

matrix.

The target obel is subdivided as outlined in Figure 4.20 (a) through (c). The distance values needed to determine overlap can be easily updated using previous techniques because the edges and faces of the children are parallel to those of the parent. If the relative tree level difference between the target obel and overlay obels is fixed, the number and arrangement of overlay obels in the overlay cannot be fixed unless some restrictions are placed on the matrix coefficient values. An alternate strategy that has been successfully applied in practice is to fix the overlay configuration and to force subdivide the target obel at algorithm initialization until coverage is guaranteed.

4.6.5 Nonlinear Transformations

The parallelogram target overlay scheme can be generalized to a quadralateral in order to accomodate nonlinear transformations of the form:

$$\begin{aligned}x' &= A_1x + B_1y + C_1 + D_1xy \\y' &= A_2x + B_2y + C_2 + D_2xy\end{aligned}\tag{4-18}$$

Figure 4.21(a) shows the required target shape. Reverse flow from target to overlay is again assumed.

Calculation of the x' value is illustrated in Figure 4.21(b). The C_1 and A_1x terms are as before. When the value of y is considered, however, it can be noted that the shift

to the right due to y will vary linearly with x over a range from 0 to B_1 at the left side ($x=0$) to 0 to $B_1 + D_1$ at the right side ($x=1$). The y component is thus $y(B_1 + D_1x)$ or the remaining required two terms of $B_1y + D_1xy$.

In 3-D the transformations can be expressed as follows:

$$\begin{aligned}x' &= A_1x + B_1y + C_1z + D_1 + E_1xy + F_1xz + G_1yz \\y' &= A_2x + B_2y + C_2z + D_2 + E_2xy + F_2xz + G_2yz \\z' &= A_3x + B_3y + C_3z + D_3 + E_3xy + F_3xz + G_3yz\end{aligned}\tag{4-19}$$

The locations of the vertex points of the target obel are shown in Figure 4.22. The coefficient subscripts from (4-19) have been deleted for clarity. The subscripts are simply the directional displacement for the indicated vertex points ($1=x'$, $2=y'$, $3=z'$). The origin, for example, is located at $x'=D_1$, $y'=D_2$, $z'=D_3$.

The location of the vertex points of the children of the target obel can be calculated using simple arithmetic as noted in Figure 4.23 for 2-D. A problem arises, however, when determining intersection. The child edges and faces are no longer guaranteed to be parallel to the equivalent parent edges and faces. Except for special cases, intersection has not been found to be easily determined using only arithmetic.

A special case is shown in Figure 4.24. The left and right edges are parallel to the y axis. The equivalent edges in its children will also be parallel to the y axis. A new

scheme is employed to determine if an overlay obel vertex point is on the positive or negative side of an edge line. For each overlay the intersection point between the target edge lines and vertical lines through overlay obel vertices is maintained. In Figure 4.24 points A and B are two such points. The location of the vertex point relative to the intersection points determines its half-plane. Because the vertical line is parallel to the left and right edges, the new intersection point for a child, point C in the diagram, is simply the average of the two parent values. The only remaining problem is computing the intersections for vertical lines through the new vertex points created when the overlay obels are subdivided. Because the new vertical lines will be half way between the parent lines, the new intersections are also the averages of the parent values.

Another problem arises in 3-D. In the linear case, the faces of the target obel are determined by 3 points in space and are therefore co-planar. In the nonlinear case, more degrees of freedom are available; all four points can be located independantly. The face is not necessarily planar. It may be a ruled surface between two segments of lines in space.

4.6.6 Perspective Transformation

In order to avoid the division required in generating a perspective display, the objects can be deformed so as to appear as if in a perspective view when an orthographic projection is used.

For the situation in Figure 4.25(a), an object with a height of y at a location x units from the observer should be "stretched" to a height of y' units according to the following:

$$y'/x_s = y/x \text{ or } y' = yx_s/x \quad (4-20)$$

For simplicity, let $x_s = 1$. Thus, y' reduces to:

$$y' = y/x \quad (4-21)$$

The triangular shaped target obel shown in Figure 4.25(b) will perform this transformation. The value of y' is the fraction of the vertical distance from the base of the target obel to the upper edge of the target obel at which the point (x,y) resides. It is y divided by the vertical distance. For this target shape it is equal to x . In the target universe the y' value is thus measured as y/x as required for a perspective deformation.

A more general situation is outlined in Figure 4.26(a). The viewpoint is located at (x_v, y_v) relative to the origin of the object universe. The screen is located at an x value of x_v . The universe edge size is e . A point (x,y) will be

transformed according to the following relation:

$$x' = x$$

$$(y' - y_v) / (x_s - x_v) = (y - y_v) / (x - x_v)$$

or

$$y' = y_v + (x_s - x_v)(y - y_v) / (x - x_v) \quad (4-22)$$

An overlay to perform this transformation is presented in Figure 4.26(b). The distances are as follows:

$$\begin{aligned} a &= y_v [1 + x_v / (x_s - x_v)] \\ b &= a - ey_v / (x_s - x_v) \\ c &= y_v - x_v(e - y_v) / (x_s - x_v) \\ d &= c + e(e - y_v) / (x_s - x_v) \end{aligned} \quad (4-23)$$

A point (x, y) in the overlay will be transformed into a point (x', y') in the universe of the target as follows. As the value of x changes from 0 to e , the u value of the bottom edge of the target will vary linearly from a to b . It can thus be expressed as:

$$a - (x/e)e(y_v / (x_s - x_v)) = a - xy_v / (x_s - x_v) \quad (4-24)$$

The top target edge can likewise be found to be:

$$c + x(e - y_v) / (x_s - x_v) \quad (4-25)$$

The height in the overlay universe of the target as a function of x is the top edge value minus the bottom or:

$$[c + x(e - y_v) / (x_s - x_v)] - [a - xy_v / (x_s - x_v)]$$

which, after substitution for the values of c and a reduces to:

$$e(x-x_v)/(x_s-x_v) \quad (4-26)$$

The value of y above the bottom edge is:

$$y - [a - xy_v/(x_s-x_v)] \quad (4-27)$$

The transformed value will be this vertical distance divided by the vertical height of the target for this value of x times the size of the universe (e) or:

$$\begin{aligned} y' &= ((y - (a - xy_v/(x_s-x_v))) / (e(x-x_v)/(x_s-x_v)))e \\ &= y_v + (x_s-x_v)(y-y_v)/(x-x_v) \end{aligned} \quad (4-28)$$

which matches the required transformation from (4-22).

An interesting property of the transformation is that only those sections of the old universe which will fall within the new universe and therefore (presumably) the field of vision of the observer, are examined in detail. Parts which clearly will fall outside the transformed universe are eliminated at the upper levels. Again, this is a benefit of spatial pre-sorting.

It should be noted that in practice, it may very well be more advantageous to provide a division operation capability within the display algorithm and directly perform a perspective transformation rather than compute the deformed object for each display frame.

4.7 Swept Volume

In applications such as NC tape verification and collision avoidance as a part of trajectory planning in robotics systems, the "object" formed by the volume swept out by an object in motion is of major importance. An octree swept volume algorithm should be able to accept an object in octree format together with a curve in space and generate the swept object in octree format in a single pass.

The following is the "rotational" swept volume algorithm. The object is rotated in a fixed body movement about a line in space parallel to an axis.

4.7.1 Rotational Swept Volume

A typical rotational sweep is shown in Figure 4.27. It is assumed that an arbitrary angle of rotation can be specified; the angle is not restricted to discrete values. Most 3-D problems can be handled as an extended 2-D case about a vertical axis.

An immediate difference vis-a-vis previous situations where obel locations are computed within an algorithm, is the change in sweep distance for an obel as a function of its distance from the axis of rotation. It is also obvious that the input obels and output obels are no longer in perfect alignment. One is rotated with respect to the other as the

sweep progresses. A multiple obel overlay approach will therefore be employed.

The strategy is outlined in Figure 4.28. In 2-D, a list overlays each containing 9 obels will be maintained for each output obel. The center obel of the overlay will be one which intersects the center of the output obel. The center of output obel A, for example, is within the sweep of the center overlay obel. It is therefore added to the list. As shown by output obel B, all neighbors which could intersect are included. Clearly, if all such overlays are in the list, all input obels having a sweep that could intersect the output obel, are examined. Obels in one overlay may, of course, appear in other overlays. For this reason, the overlays will contain pointers to a common location for a particular node and its associated information in order to conserve storage and to prevent multiple computation of the same information.

If any overlay is found to completely cover the output obel with swept input obels with the value F, the output node is marked F. All attached overlays can then be deleted. If all intersecting nodes are found to be empty (E), the overlay is deleted. If a final determination cannot be made on the basis of the overlays in the list, they are subdivided and the appropriate child overlays attached to the children of the output node for consideration at the next lower level.

The most difficult part of the operation is determining

the intersection situation between the output obel and the swept overlay. The situations are divided into two classes, those not involving the ends of the sweep and those that do. The former is considered first.

An obel swept around an axis sweeps out a hollow cylinder or a part of one as shown in Figure 4.29(a). The two vertex points which are at the most distant and closest points sweep the edges of the cylinder. Their starting distances from the center of rotation, R_1 and R_2 are the exterior and interior radius values, respectively. The cylinder is essentially swept out by a line between these two vertices (neglecting endpoints). Note that the orientation of the obel changes as it moves. The two vertex points are determined by the starting quadrant relative to the center of rotation as shown in Figure 4.29(b).

As shown in Example 4.3 above, calculating the distance squared from the center of rotation for each vertex of an obel can be calculated using simple arithmetic from parent values. The same is true for the output obels. In 3-D, this is also true if the center of rotation is parallel to an axis. In fixed situations it might be possible to so orient the coordinate system. Otherwise, the object can be rotated into position or more complex arithmetic can be allowed.

Some of the obel intersection techniques developed for octree object generation above will be employed. Most require that the test obel be guaranteed to be within a

section containing surface point normals from a single direction octant. To simplify matters, it will be assumed that the axis of rotation will be through the center of the universe. This will force all obels (except the root) to be within a restricted region.

For a single band, the output obel intersection test is illustrated in Figure 4.30. Two convex objects are involved, section B, the area below the inner edge, and A + B, the area below the outer edge. If the negative test vertex for the direction octant is beyond the outer edge, there can be no intersection. This case is $R_{\min}^2 \geq R_1^2$. If the positive test vertex is within B ($R_{\max}^2 \leq R_2^2$), the obel is entirely within the B section, and no intersection with the band can exist. If the negative test vertex is beyond the inner edge ($R_{\min}^2 \leq R_2^2$), and the positive test vertex is within the outer edge ($R_{\max}^2 \leq R_1^2$), the obel is entirely within the band. Otherwise there is partial intersection.

It is, in general, unreasonable to expect that an output obel will be entirely enclosed by a band caused by a single obel. Thus the need for a multiple obel overlay. The procedure is to analyze the bands of the 9 overlay obels. They will typically overlap. They are simply consolidated into disjoint bands of solid and empty. If the obel is entirely enclosed by a solid band, an F value can be given. If it does not intersect a solid band, this overlay is discontinued. If a partial intersection exists, a P value is

issued and the overlay subdivided.

The second class of intersection, those involving the endpoints of the sweep, can be analyzed in a number of ways. The methods developed above to determine the intersection between a test obel and an object can be utilized. The test obel is the output obel and the object is the swept overlay.

An obel can be eliminated from consideration for intersection if it is outside the bounding box or is beyond or entirely within the radius values of the sweep. After this, the tests not involving end conditions can be easily identified. In Figure 4.31 they are the obels found to be entirely within the region shown above and to the right of the dotted lines. The remaining obels require more extensive testing.

As shown by the case in Figure 4.31, the vicinity of the starting location can be divided into two parts, the triangles formed by vertex points 1, 3 and 4, and the sweep of the line between vertices 1 and 3 rightward. Note that the swept line always starts out at a 45 deg. angle. At the ending location, the sweep ends with a rotated square. It can be divided into two parts, the triangle formed by points 1, 2 and 3, and the leftward sweep of line segment 1 to 3. In all cases the 1 to 3 diagonal line will be at a particular angle, a displacement from 45 deg. of the sweep angle. Information about a line at this angle can be precomputed.

The location of vertex points at the end location can be

easily computed using simple arithmetic from the the starting points. The values A_x , A_y , B_x and B_y are maintained for each point where x and y are the distance from the center of rotation, $A = \cos(T)$, $B = \sin(T)$ and T is the sweep angle. They are calculated from the parent values using (4-6). The rotated coordinates are then simply $x' = A_x - B_y$ and $y' = B_x + A_y$.

Unfortunately, it is not simple to calculate the perpendicular distance to a common point using simple arithmetic because the distance changes with radius. The determination of the side of a line upon which a test vertex of the output obel lies is probably easiest calculated by (reluctantly) allowing a single multiplication operation per test.

Briefly, the previously developed intersection methods can now be used. Two tests are performed, one for the end triangle and the second for the sweep with the diagonal line used as an additional edge. One problem remains. Intersection is easily determined but enclosure may be over several sweeps and end triangles. The solution is to also check for intersection with the non-swept regions. If none is found, the output obel is enclosed.

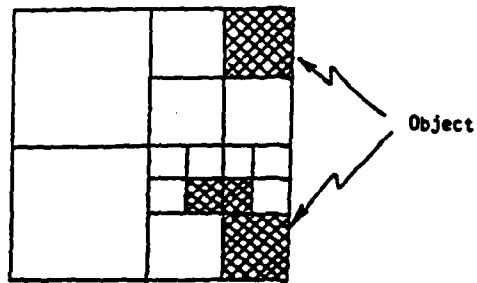
5 REFERENCES

1. Requicha, A., and Voelcker, H., "Geometric Modeling of Mechanical Parts and Machining Processes," COMPCONTROL 1979, Sopron, Hungary, Nov. 1979
2. Requicha, A., "Representations for Rigid Solids: Theory, Methods, and Systems," Computing Surveys, Vol. 12, No. 4, December 1980
3. Requicha, A., and Voelcker, H., "A Tutorial Introduction to Geometric Modelling," SIGGRAPH 1980 Tutorial, July 1980
4. Baer, A., Eastman, C., and Henrion, M., "Geometric Modeling: A Survey," Computer-Aided Design, Vol. 11, No. 5, Sept. 1979
5. Meagher, D., "Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer," Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, October 1980
6. Meagher, D., "Geometric Modeling Using Octree Encoding," Technical Report IPL-TR-81-005, Image Processing Laboratory, Rensselaer Polytechnic Institute, February 1981
7. Meagher, D., "High Speed Display of 3-D Medical Images Using Octree Encoding," IPL-TR-021, Image Processing Laboratory, Rensselaer Polytechnic Institute, Sept. 1981
8. Bell, C. G., Grason, J. and Newell, A., Designing Computers and Digital Systems, Digital Press, 1972
9. Shoor, R., "McCracken," Computerworld Extra, Sept. 1981
10. Clark, J., "Hierarchical Geometric Models for Visible Surface Algorithms," Communications of the ACM, Vol. 19, No. 10, Oct. 1976
11. Voelcker, H., and Requicha, A., "Geometric Modeling of Mechanical Parts and Processes," Computer, Dec. 1977
12. Hunter, G. M., and Steiglitz, K., "Operations on Images Using Quad Trees," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, April 1979

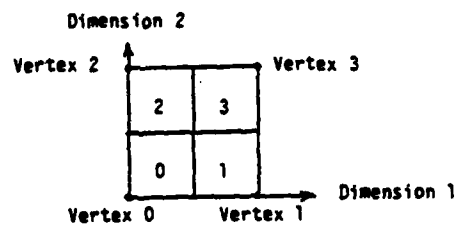
13. Samet, H., "Connected Component Labeling Using Quadtrees," TR-756, Computer Science Dept., University of Maryland, April 1979
14. Newman, W. M., and Sproull, R. F., Principles of Interactive Computer Graphics, 2nd Ed., McGraw-Hill, 1979

<u>No.</u>	<u>Test</u>	<u>Condition</u>	<u>Status</u>
1	no intersection with bounding box	no intersection	E
2	neg. test vertex on or in neg. half-space of associated face planes (any tangent plane for non-polyhedral object)	no intersection	E
3	pos. test vertex on or in pos. half-space for <u>all</u> face planes (all vertices on surface or interior for non-polyhedral)	enclosed	F

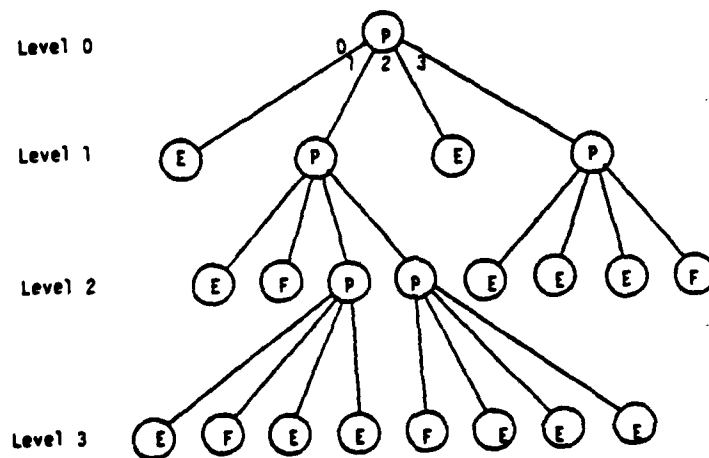
Table 4.1 - Convex Object Tests



(a) 2-D Object

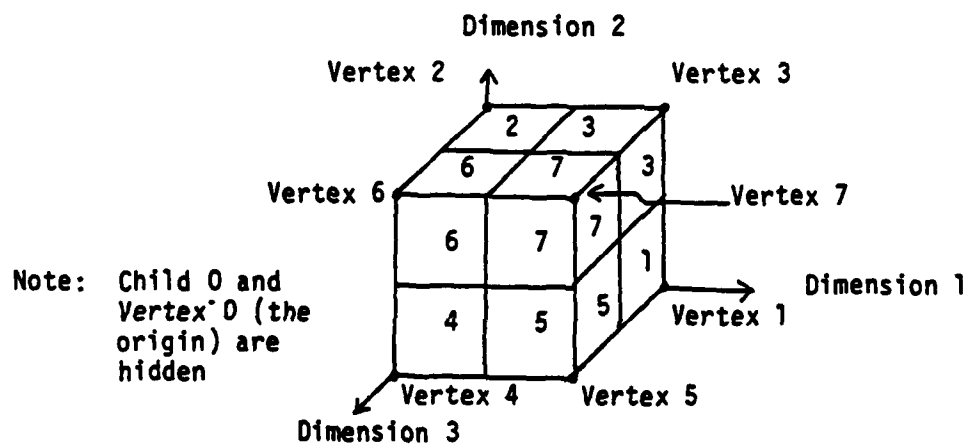


(b) Child and Vertex Labeling



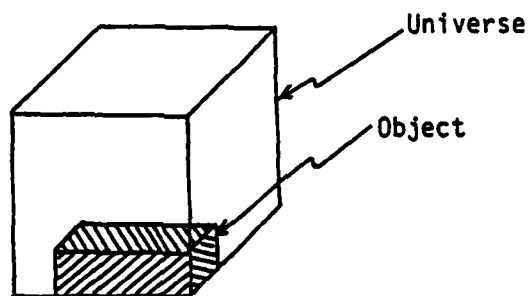
(c) Quadtree Representation of 2-D Object

Figure 3.1 Sample Quadtree

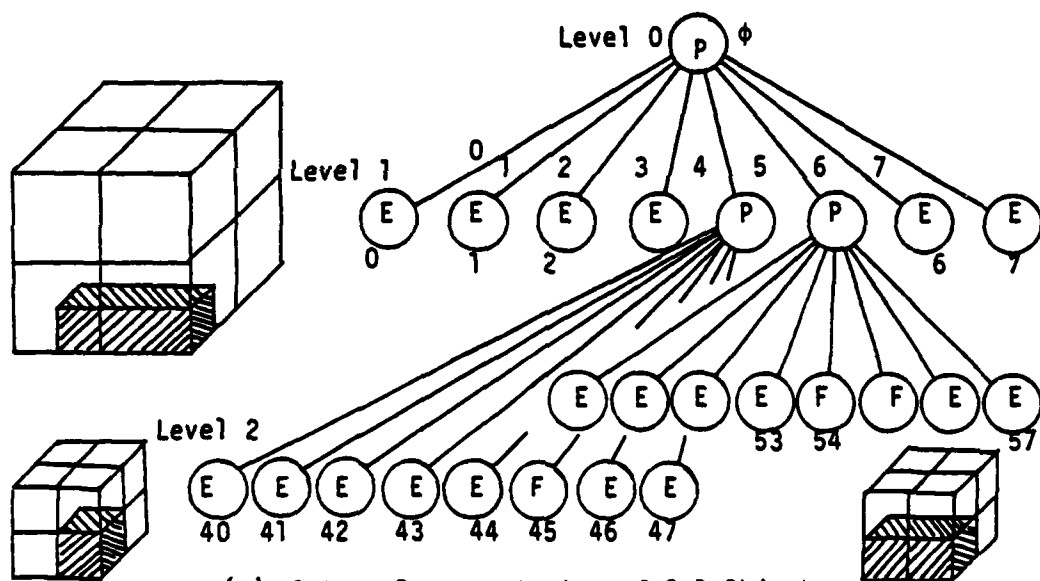


(a) Child and Vertex Labeling

Figure 3.2 Sample Octree



(b) 3-D Object



(c) Octree Representation of 3-D Object

Figure 3.2 (Continued) Sample Octree

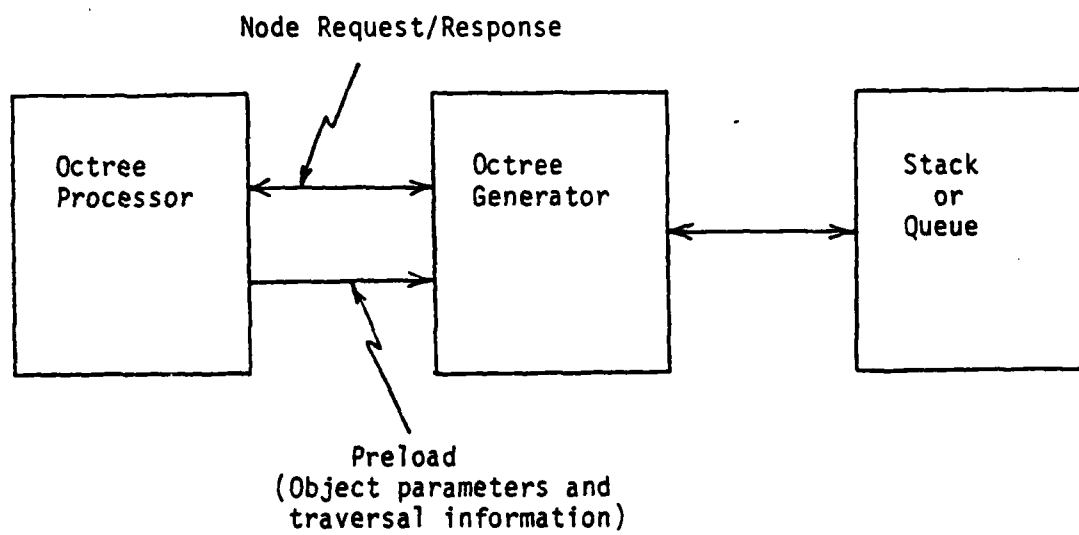


Figure 4.1 Octree Generator

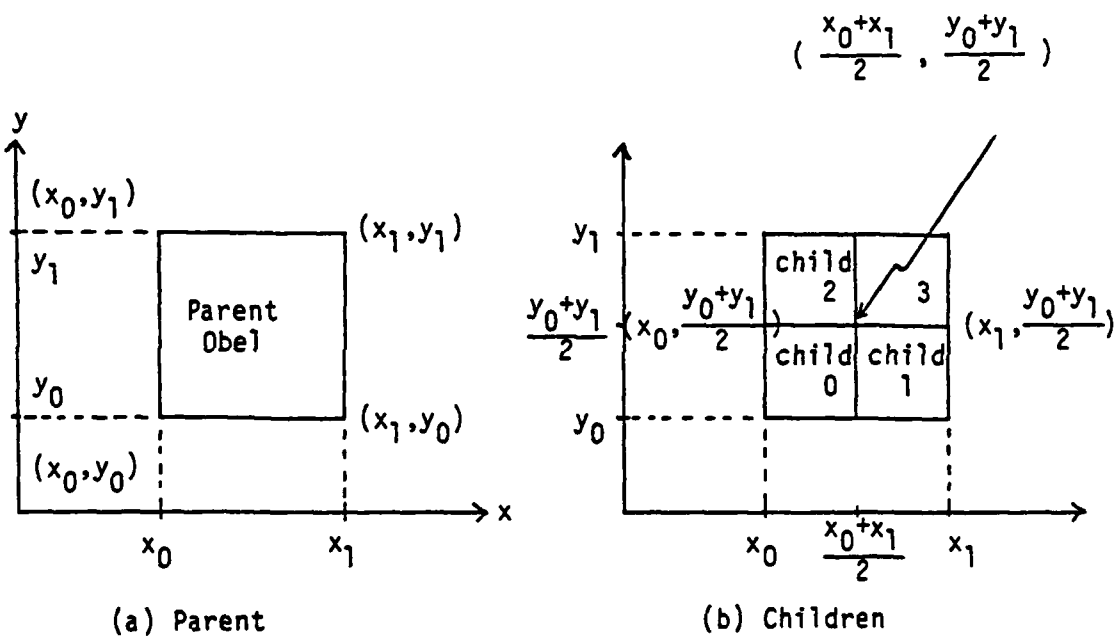


Figure 4.2 Calculation of Child Vertex Coordinates from Parent Values.

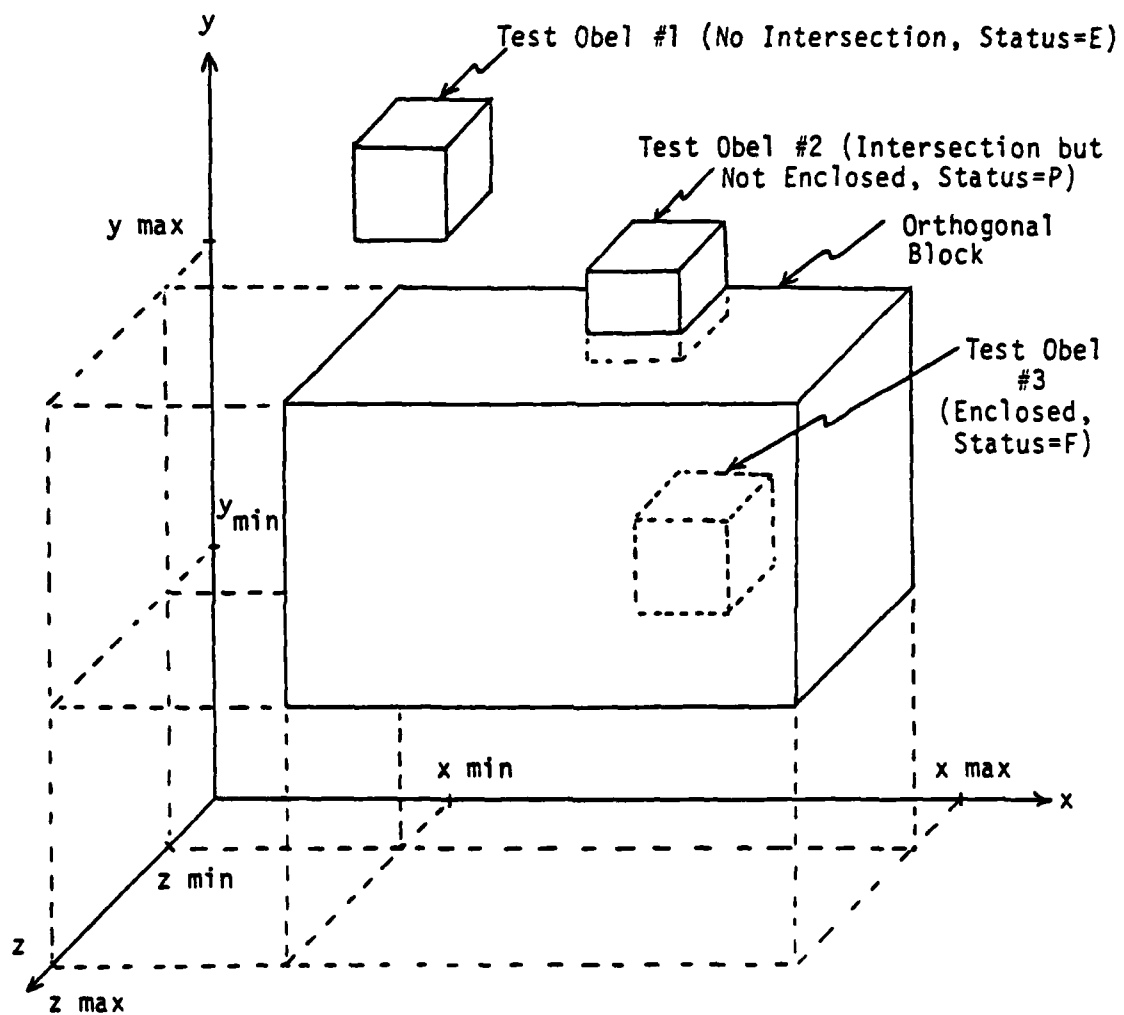


Figure 4.3 Octree Generation for 3-D Orthogonal Block

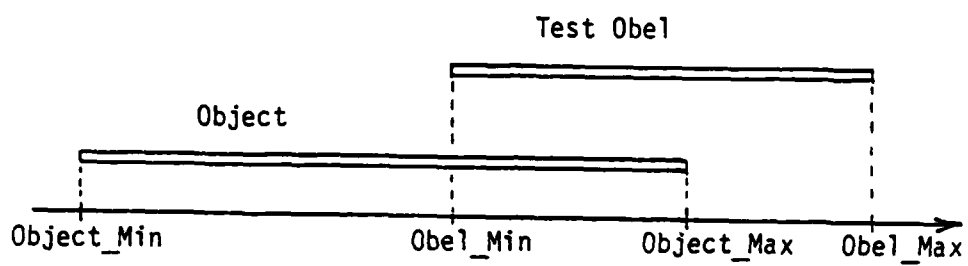


Figure 4.4 Projections of Test Obel and Object on Axis of Dimension m


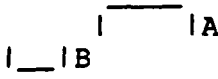

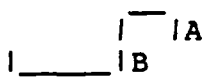
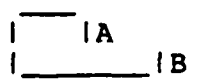
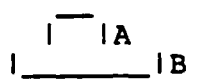

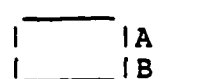
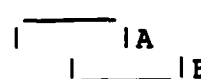
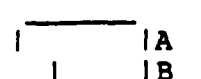
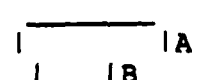

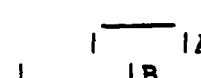
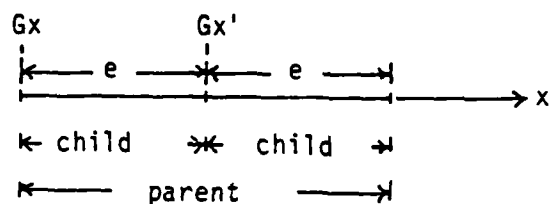
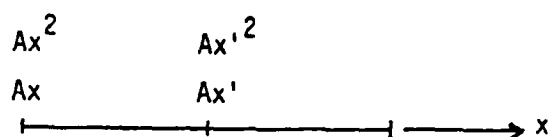
Obel-to-Object Compare						Example	Case
No.	Limits	Min:Max	Max:Min	Max:Max	Min:Min		
1	None	<	<	<	<		disjoint
2	None	>	>	>	>		disjoint
3	None	<	=	<	<		disjoint
4	None	=	>	>	>		disjoint
5	A < B	<	>	<	=		covered
6	A < B	<	>	<	>		covered
7	A < B	<	>	=	>		covered
8	A = B	<	>	=	=		covered
9	None	<	>	<	<		overlap
10	A > B	<	>	=	<		overlap
11	A > B	<	>	>	<		overlap
12	A > B	<	>	>	=		overlap
13	None	<	>	>	>		overlap

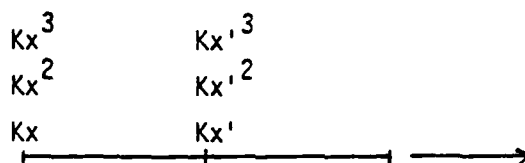
Figure 4.5 Possible Combinations of Obel and Object Projections
(with Obel_Min < Obel_Max and Object_Min < Object_Max)



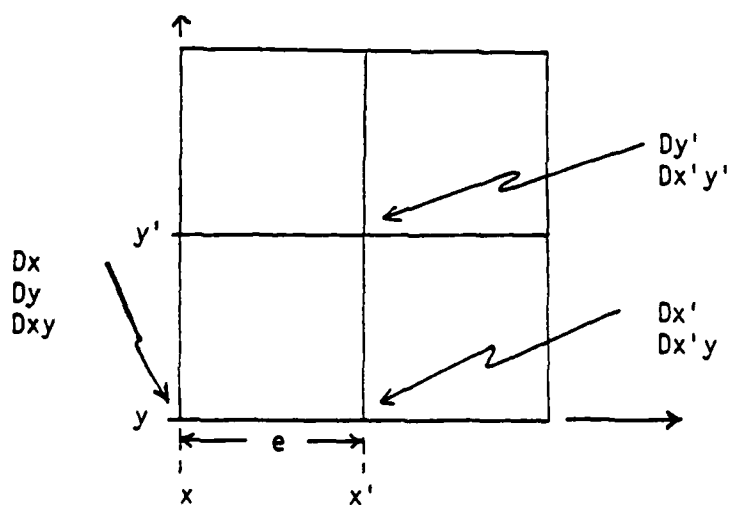
(a) Given G_x , calculate $G_{x'}$



(b) Given A_x and A_x^2 , calculate $A_{x'}$ and $A_{x'}^2$



(c) Given K_x , K_x^2 and K_x^3 , calculate $K_{x'}$, $K_{x'}^2$ and $K_{x'}^3$



(d) Given D_x , D_y and D_{xy} , Calculate $D_{x'}$, $D_{y'}$, $D_{x'y}$ and $D_{x'y'}$

Figure 4.6 Generation Terms for Child Obel from Parent Values

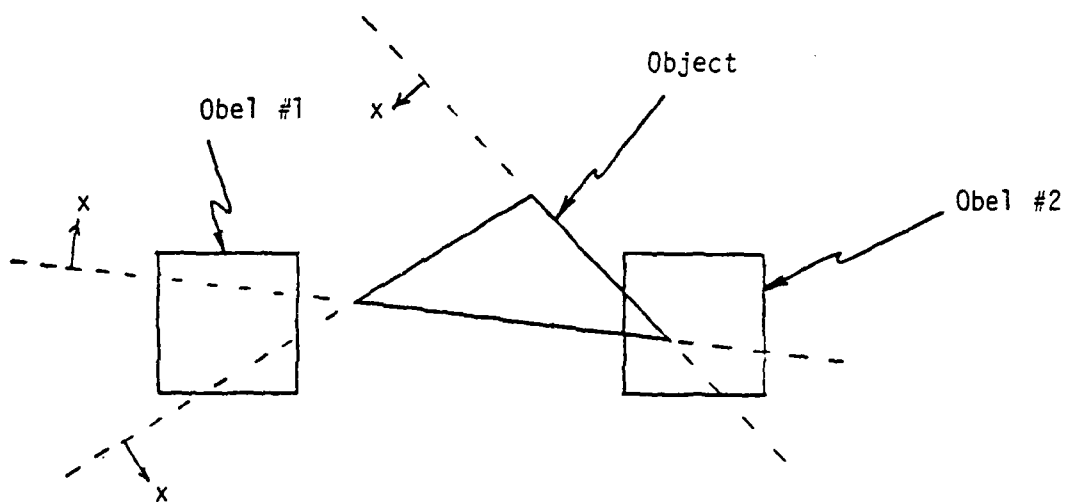
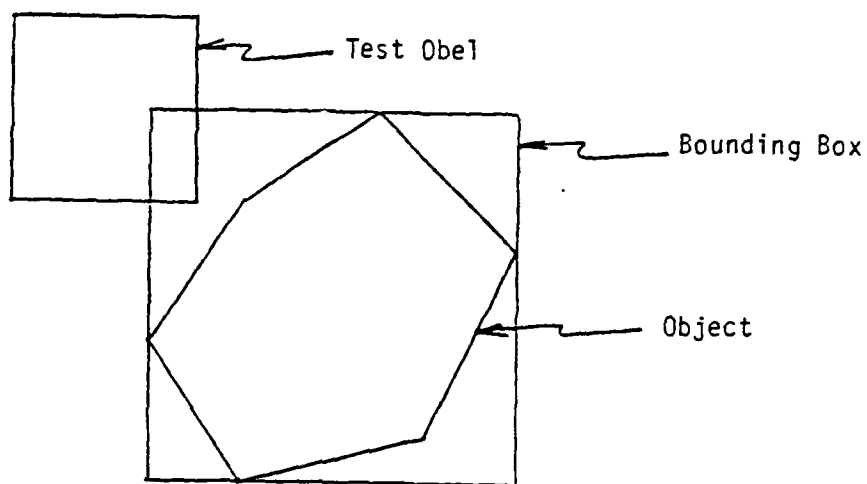
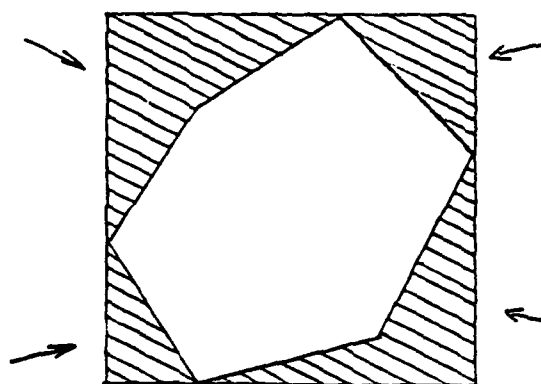


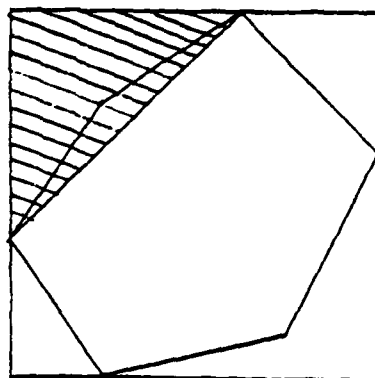
Figure 4.7 Example of Obel Status Values not Determined by Table 4.1 Tests (for both obels there is at least one vertex on positive side of every face edge).



(a) Bounding Box and Test Obel



(b) Corner Areas



(c) Corner Region

Figure 4.8 2-D Convex Polygon with Bounding Box

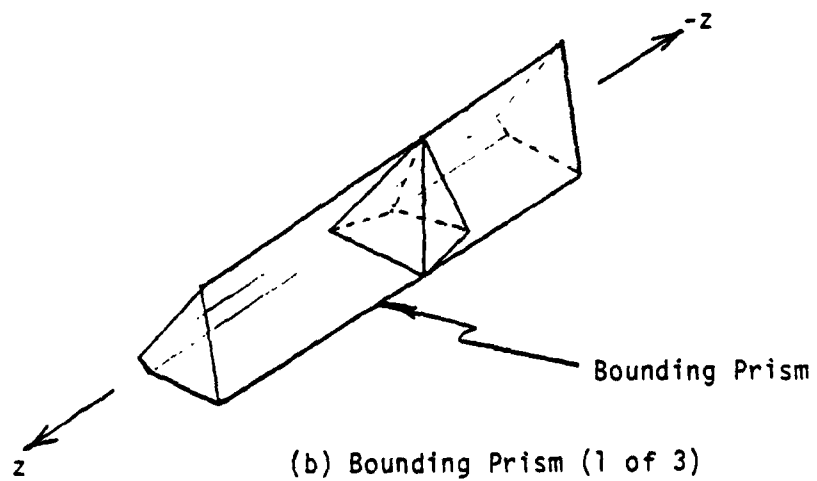
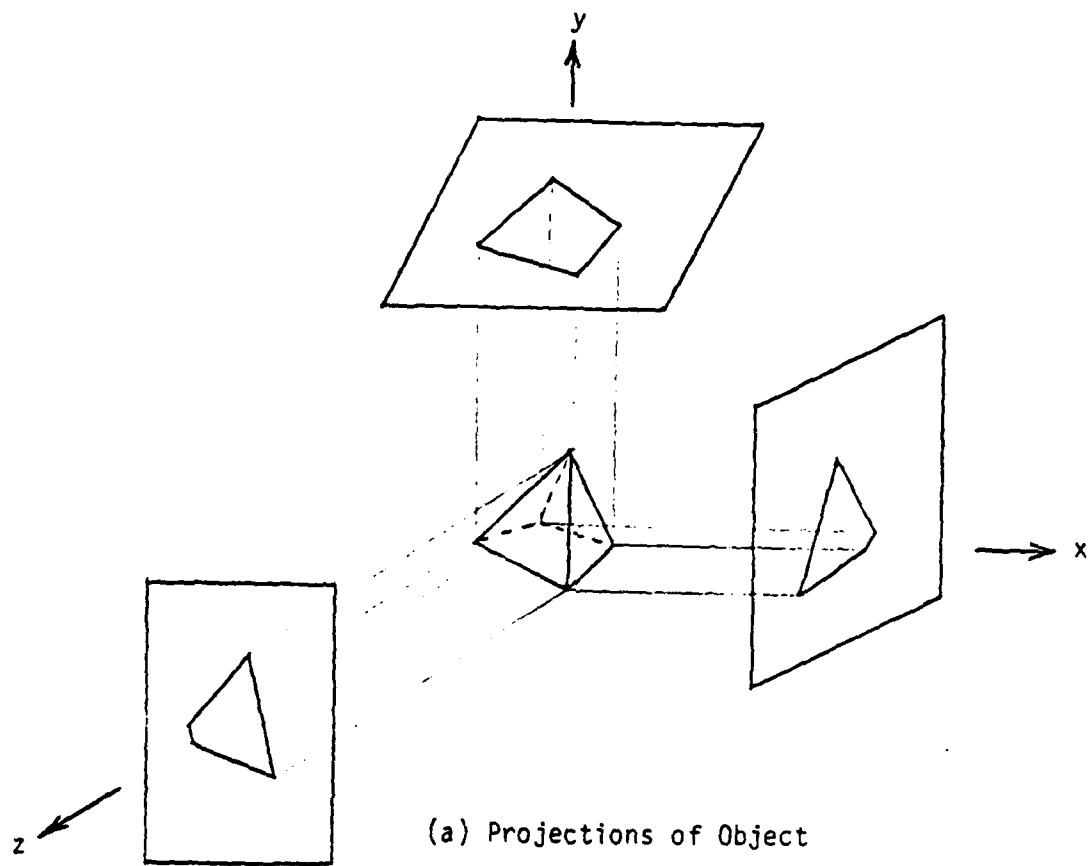
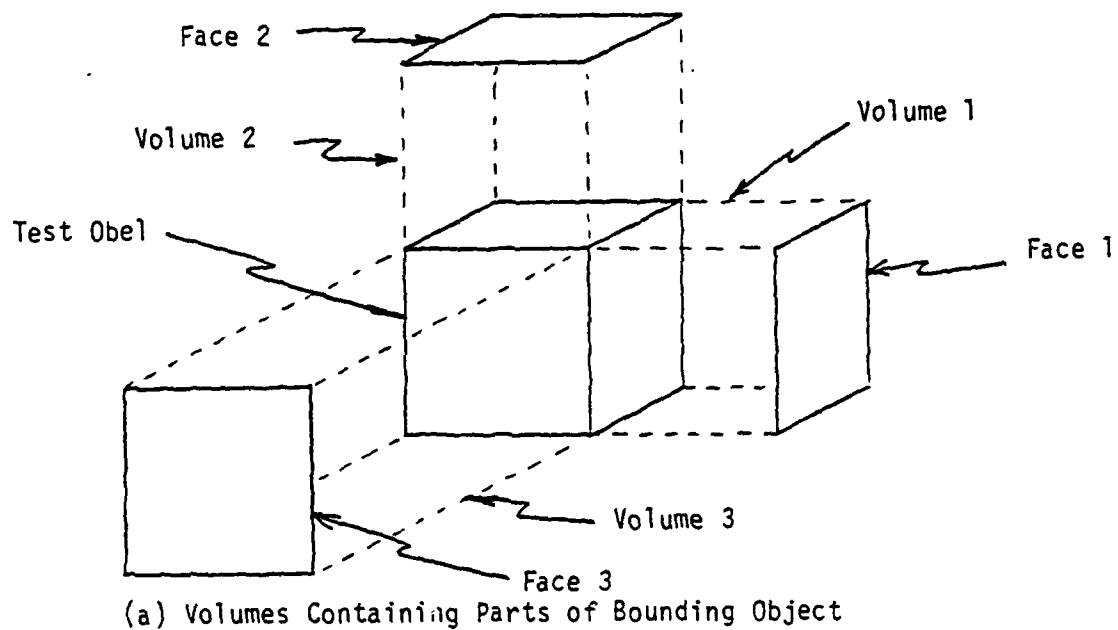
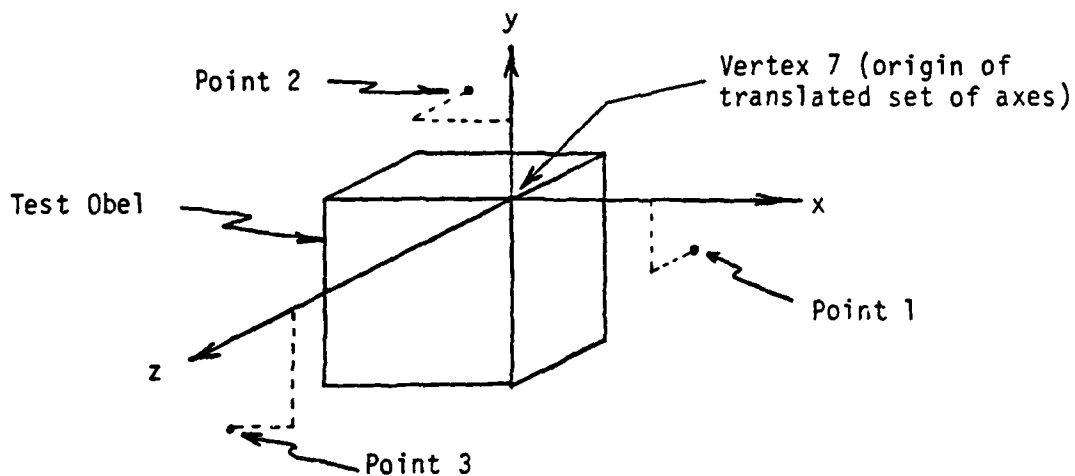


Figure 4.9 Bounding Prisms

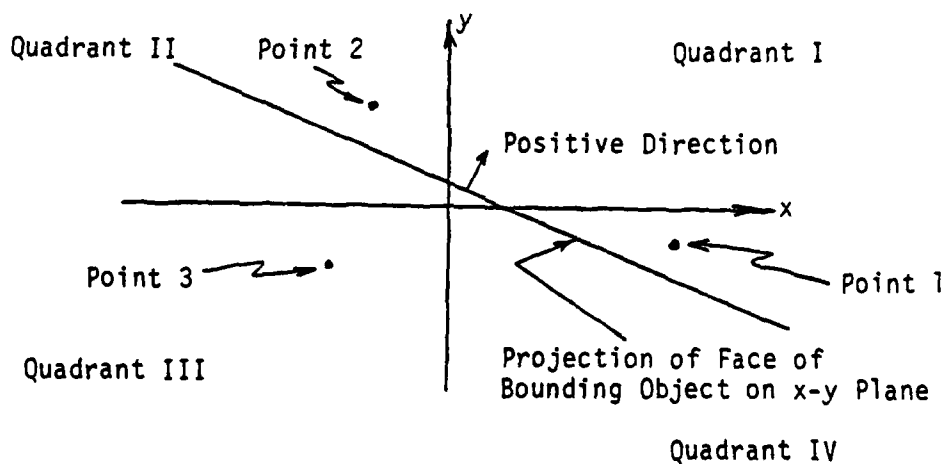


(a) Volumes Containing Parts of Bounding Object

Figure 4.10 Analysis of 3 Bounding Object Points

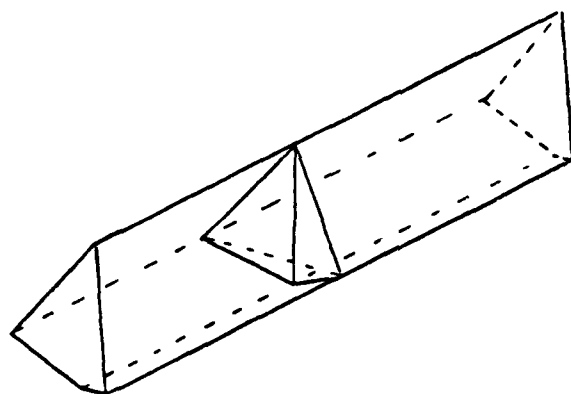


(b) Translated Set of Axes and 3 Bounding Object Points

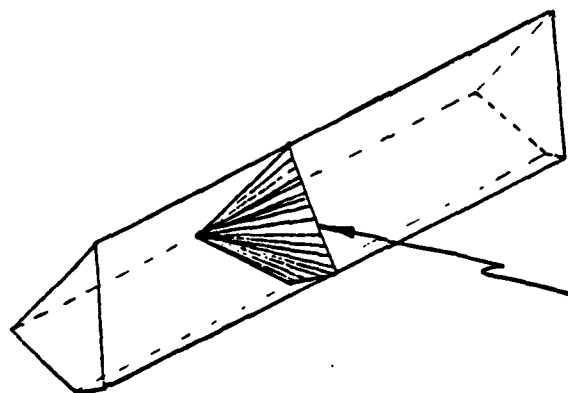


(c) Projection of 3 Bounding Object Points on x - y Plane

Figure 4.10 (Continued) Analysis of Three Bounding Object Pairs



(a) Bounding Prism



Separating Surface

(b) Two Disjoint Sections of Bounding Prism

Figure 4.11 Separation of Bounding Prism

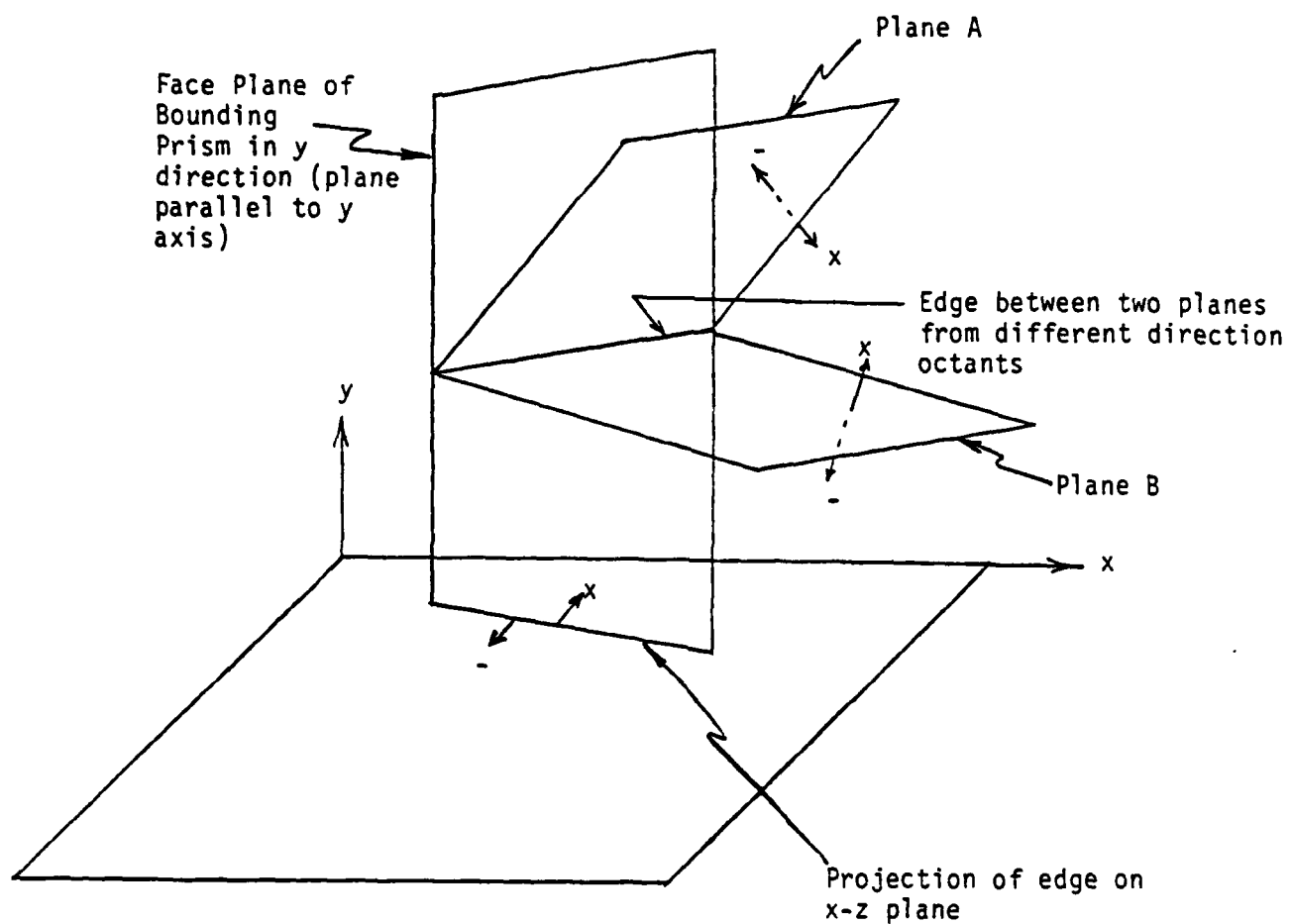
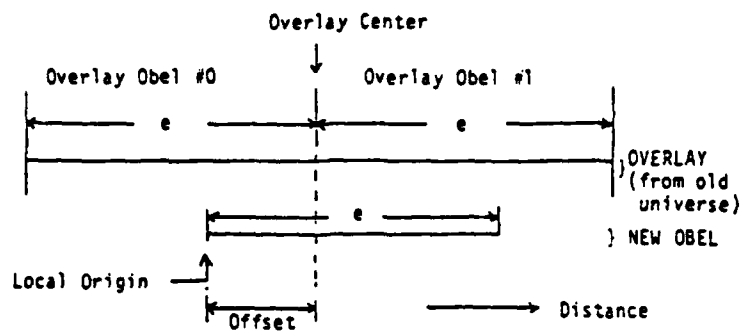
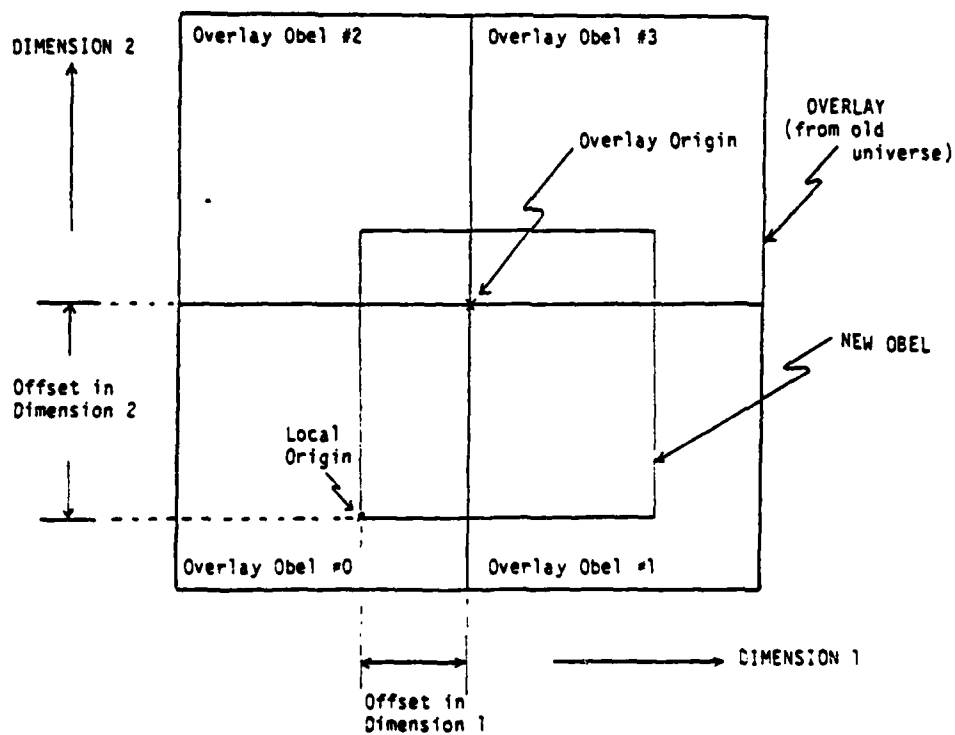


Figure 4.12 Projection of Edge Between Planes in Different Direction Octants



(a) One-Dimensional Overlay



(b) Two-Dimensional Overlay

Figure 4.13 Overlay Structure

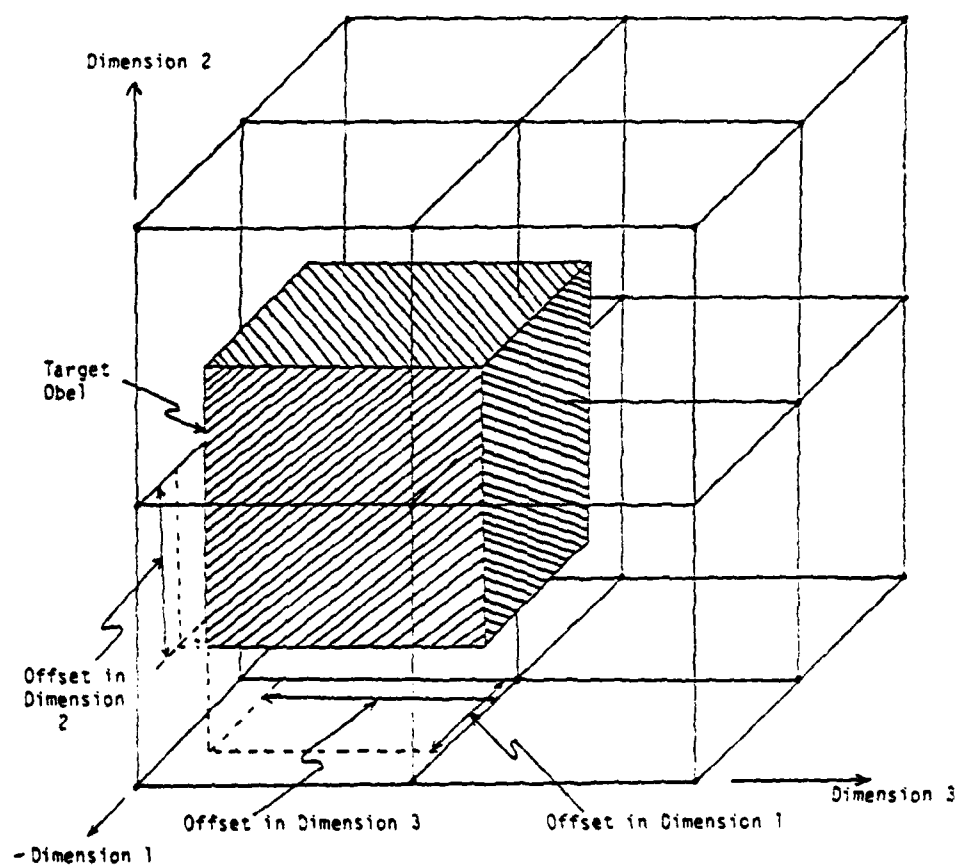


Figure 4.14 Three-Dimensional Overlay

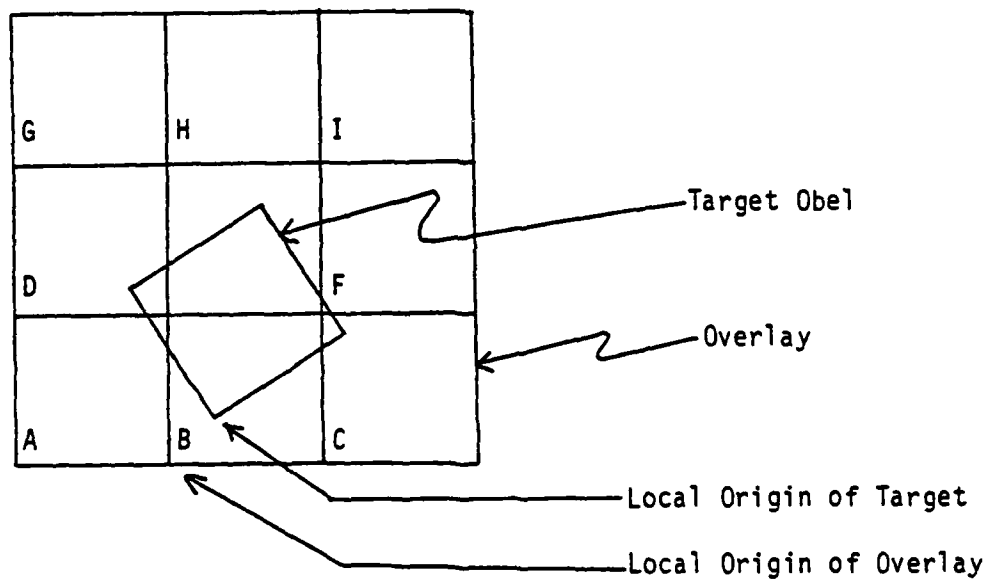


Figure 4.15 Rotation Overlay

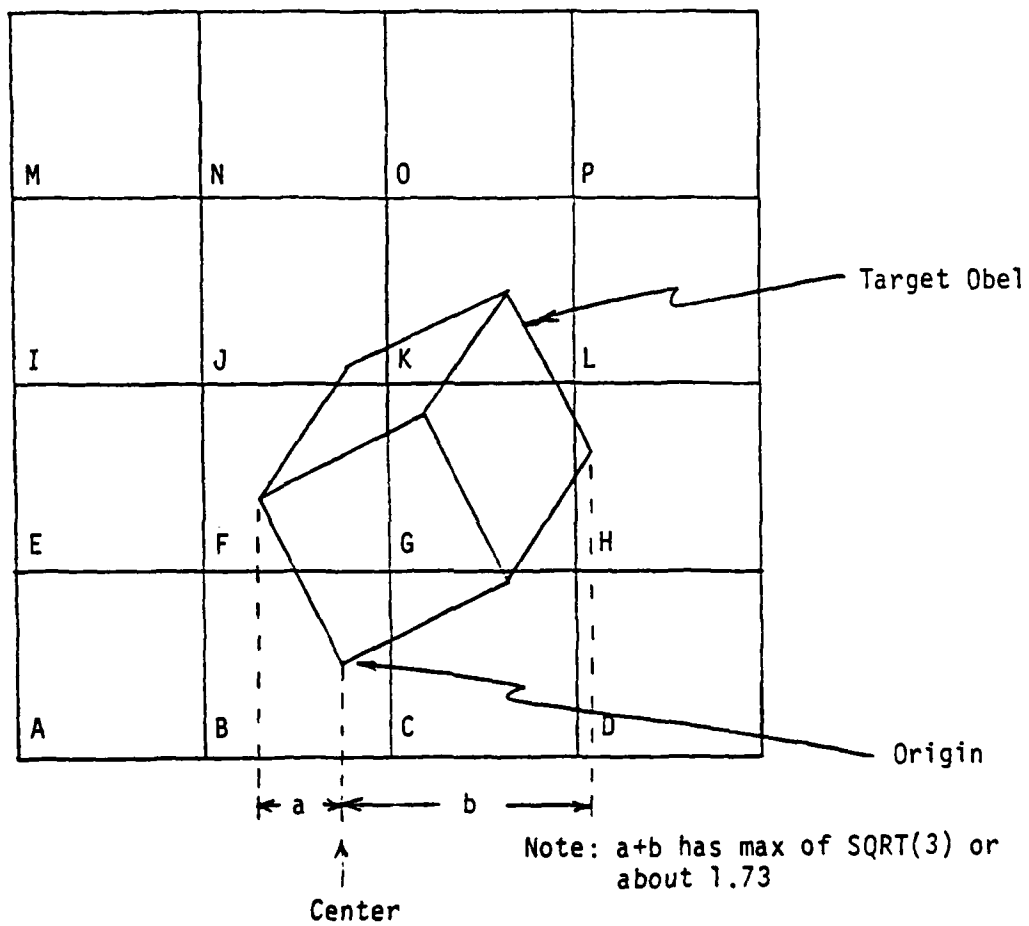


Figure 4.16 Projection of 4 by 4 by 4 Overlay for 3-D Rotation

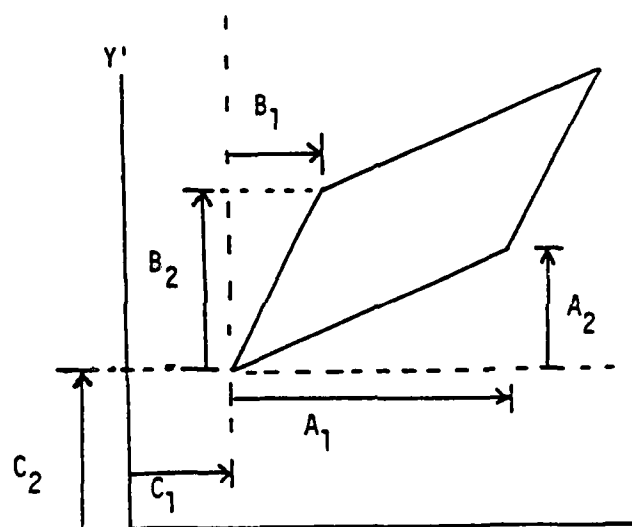
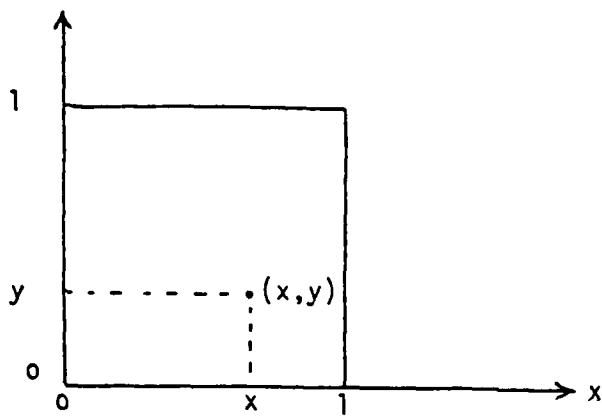
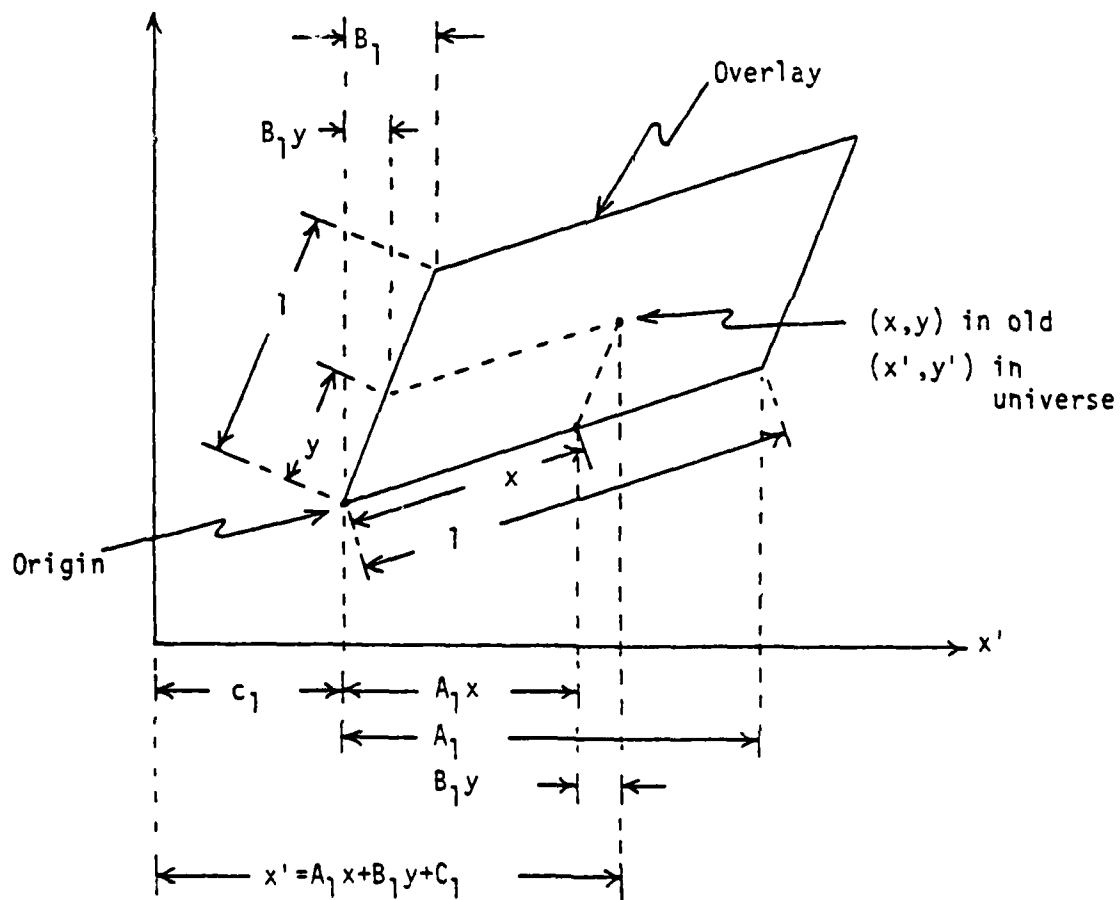


Figure 4.17 Target Obel for Concatenated Geometric Transformation

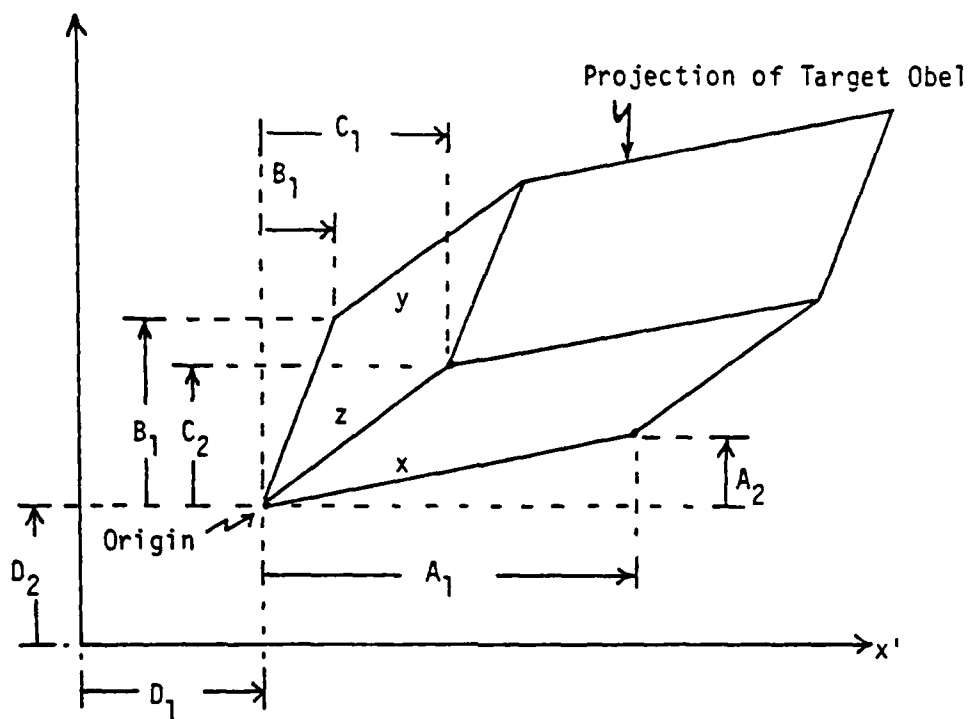


(a) Universe Before Transformation

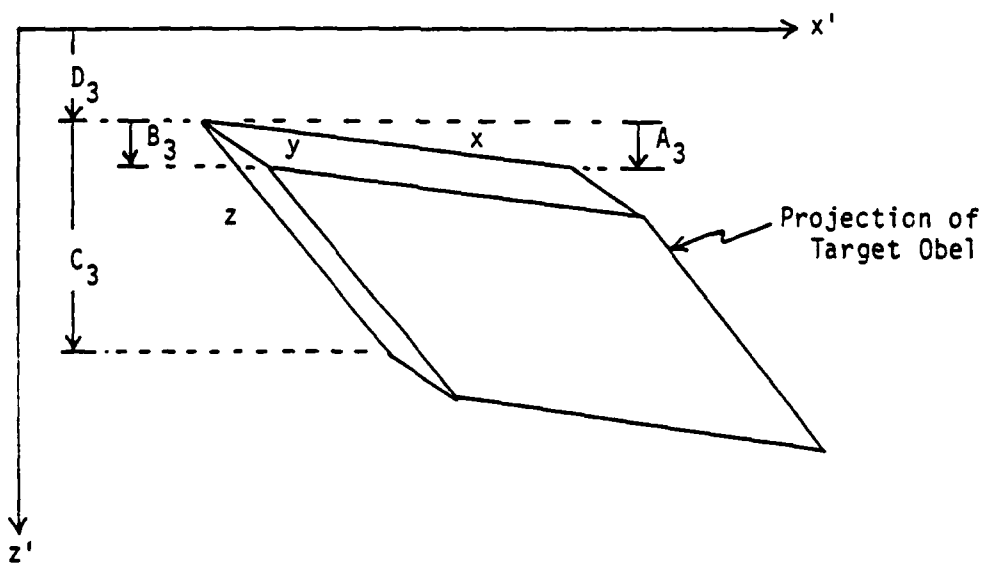


(b) Generation of Transformed x Value

Figure 4.18 Transformed Coordinate Value

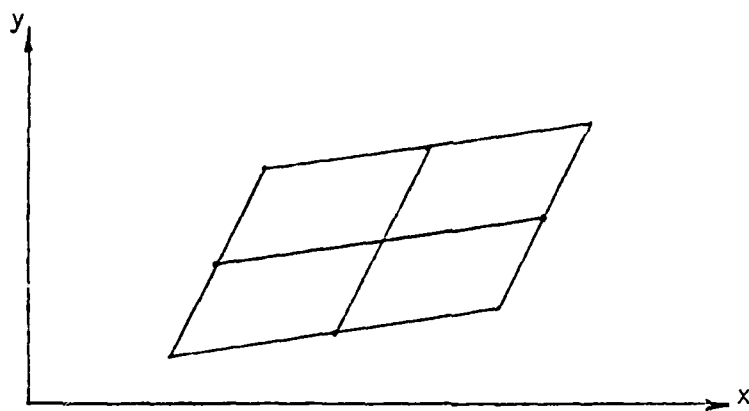


(a) Projection of Target Obel on $x'-y'$ Plane

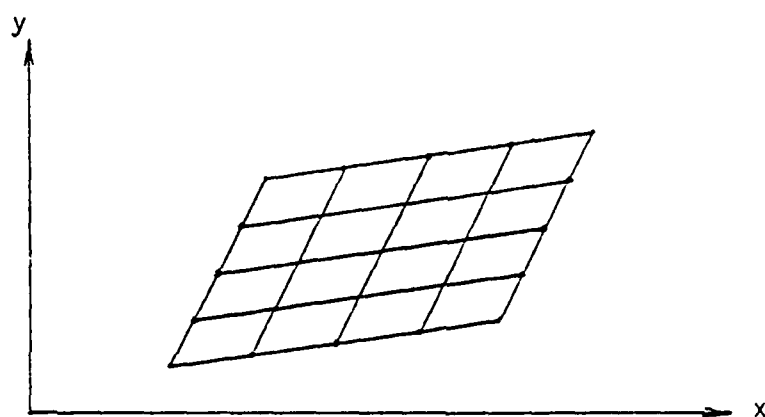


(b) Projection of Target Obel on $x'-z'$ Plane

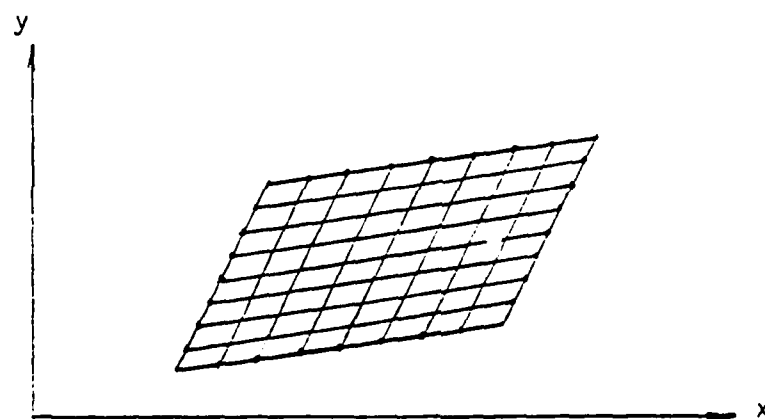
Figure 4.19 Target Obel for 3-D Concatenated Geometric Transformation



(a) Level 1 Children

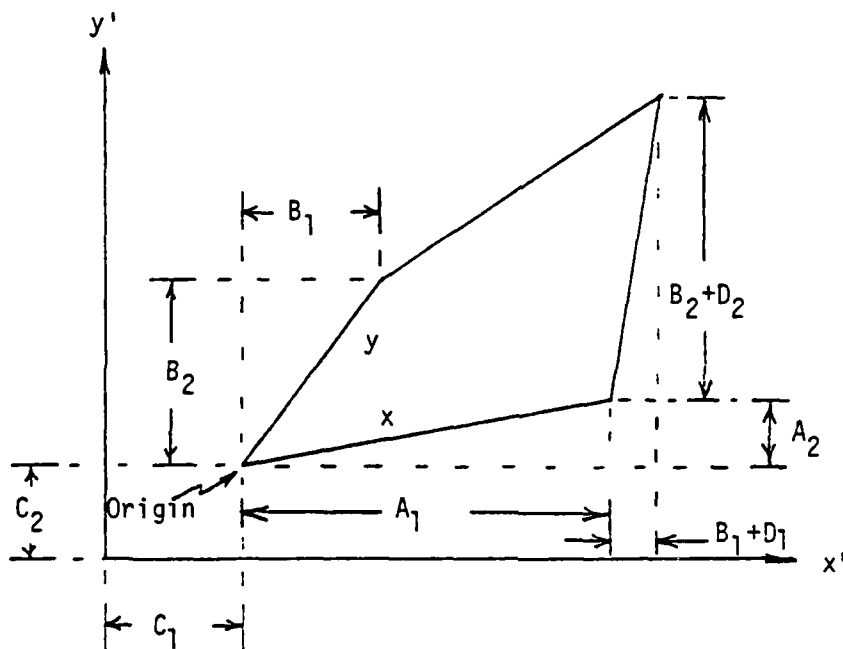


(b) Level 2 Children

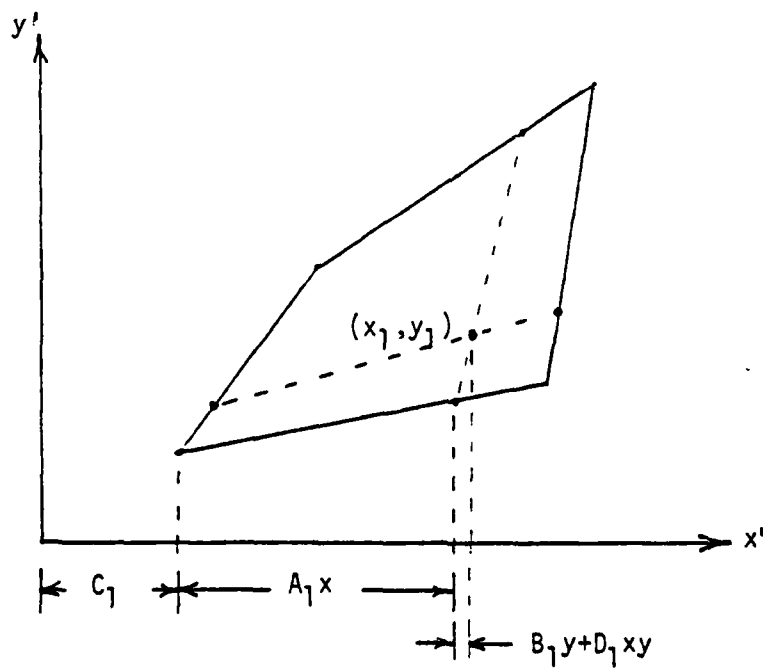


(c) Level 3 Children

Figure 4.20 Subdivision of Target Obel for Concatenated Geometric Operations



(a) Target for Nonlinear 2-D Transformation



(b) Calculation of $x' = A_1x + B_1y + C_1 + D_1xy$

Figure 4.21 Nonlinear Transformation

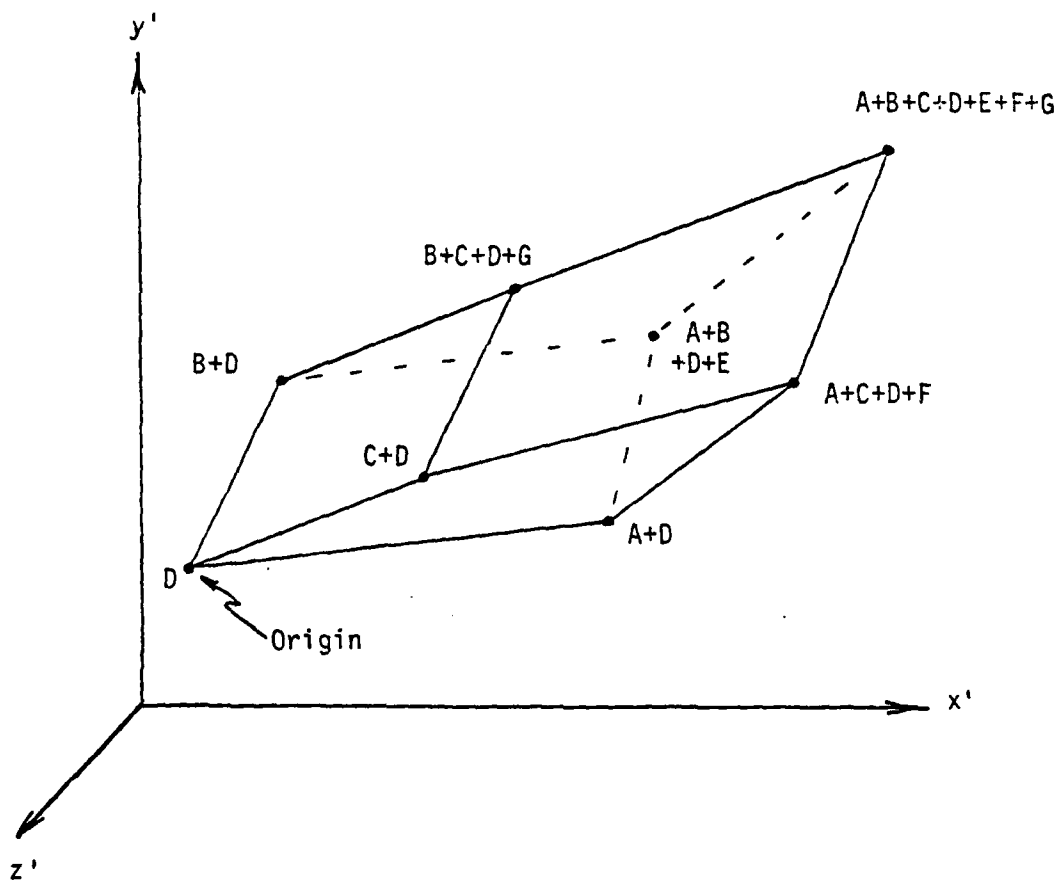


Figure 4.22 Target for Nonlinear 3-D Transformation

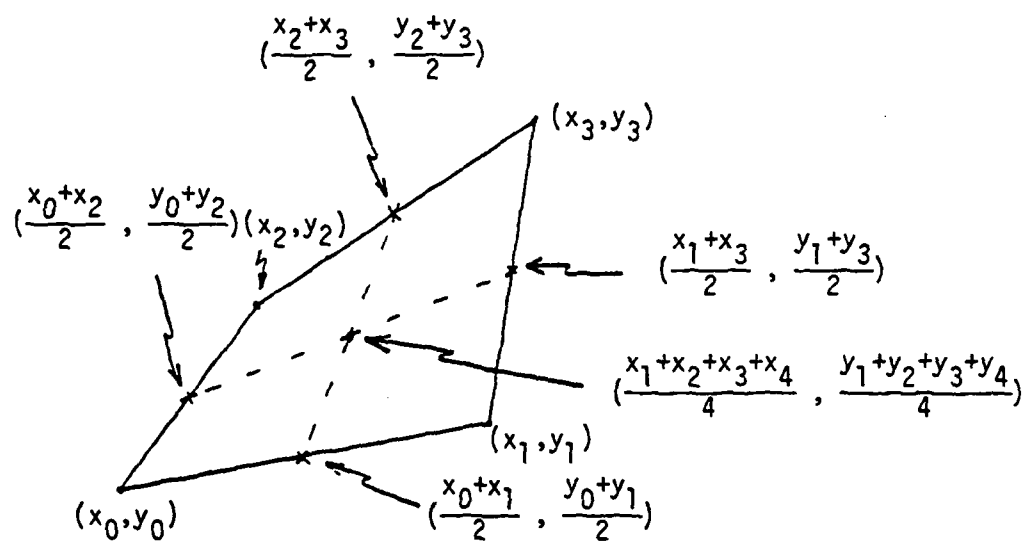


Figure 4.23 Calculation of Child Vertex Points for Quadrilateral Overlay Target

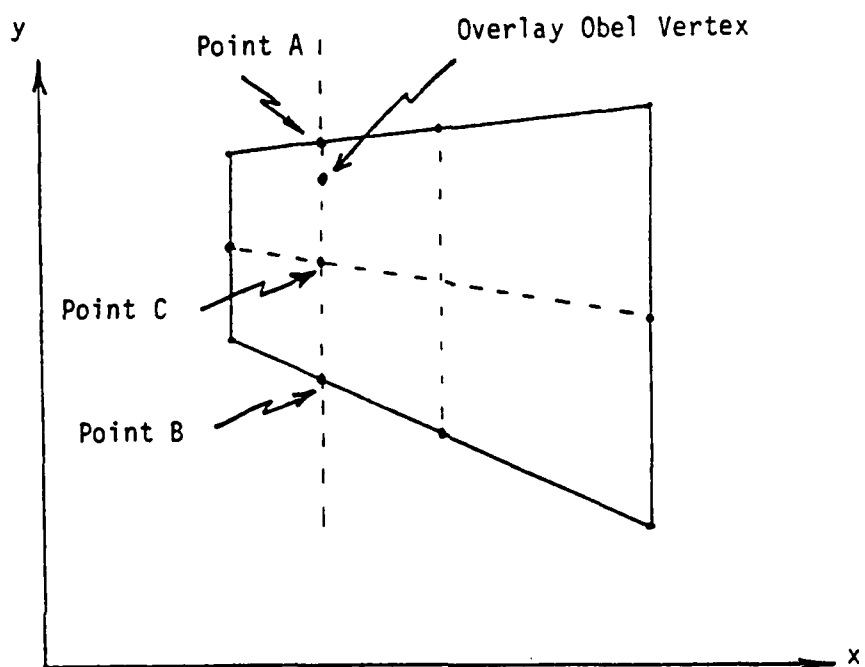


Figure 4.24 Special Case of Nonlinear Transformation Overlay Target

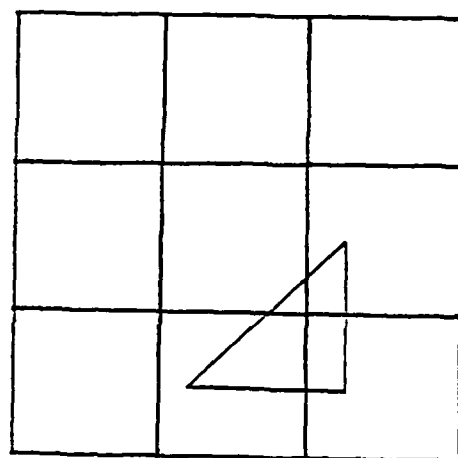
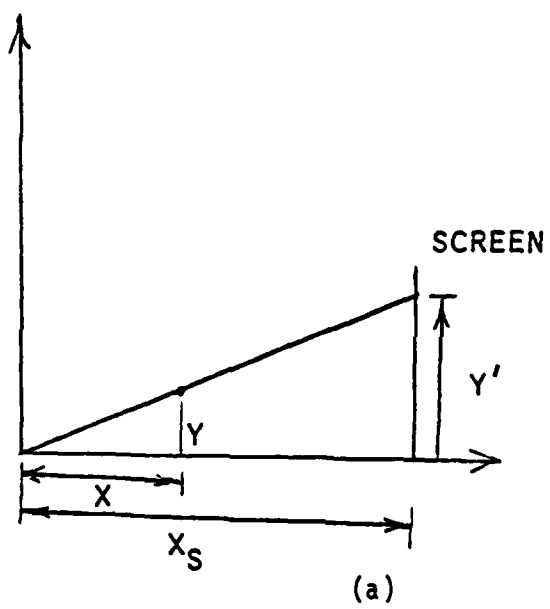
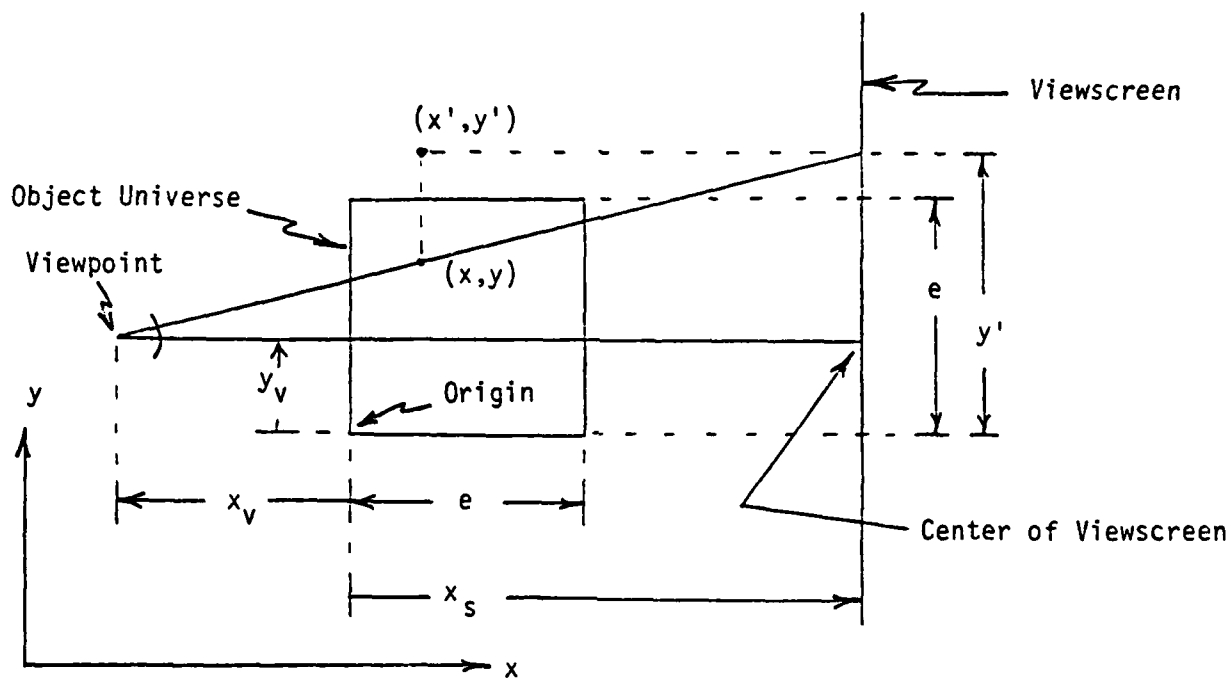
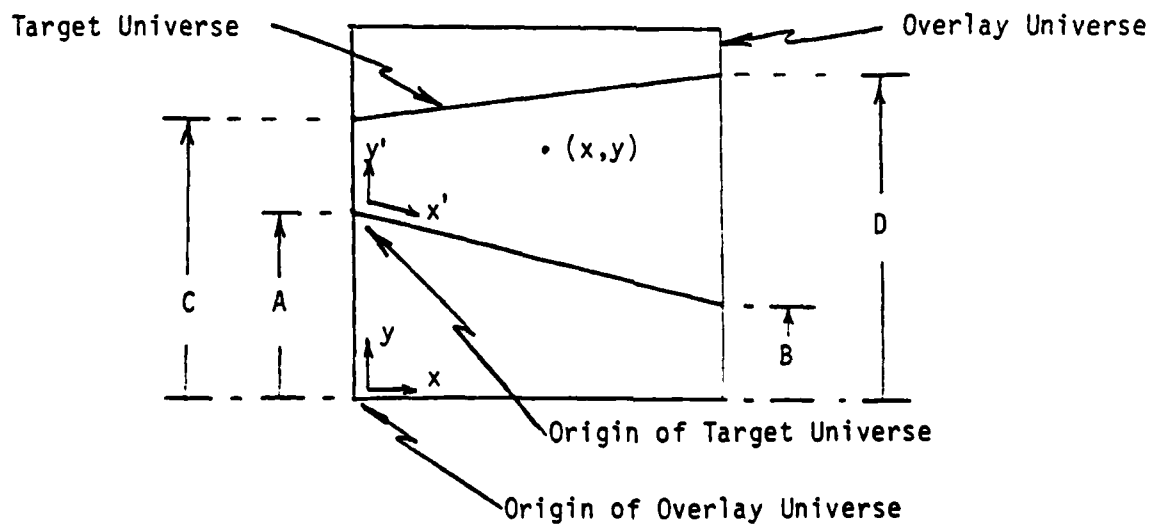


Figure 4.25 Target Obel for Perspective Transformation



(a) Transformation



(b) Target Object

Figure 4.26 Generalized Perspective Transformation

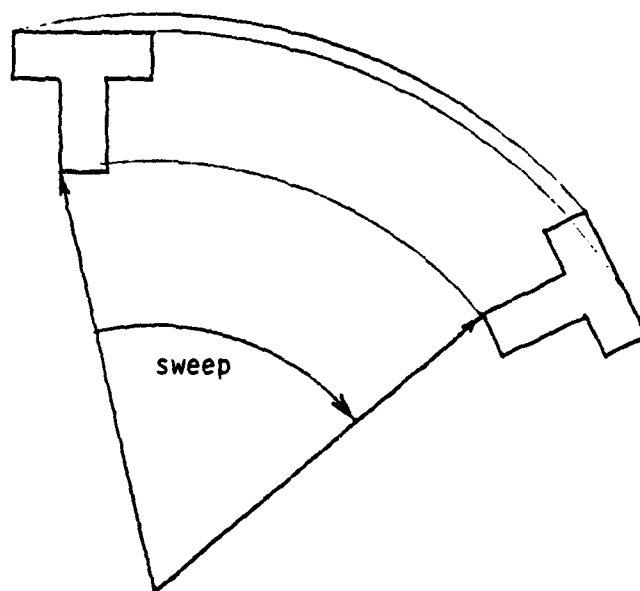


Figure 4.27 Rotational Sweep

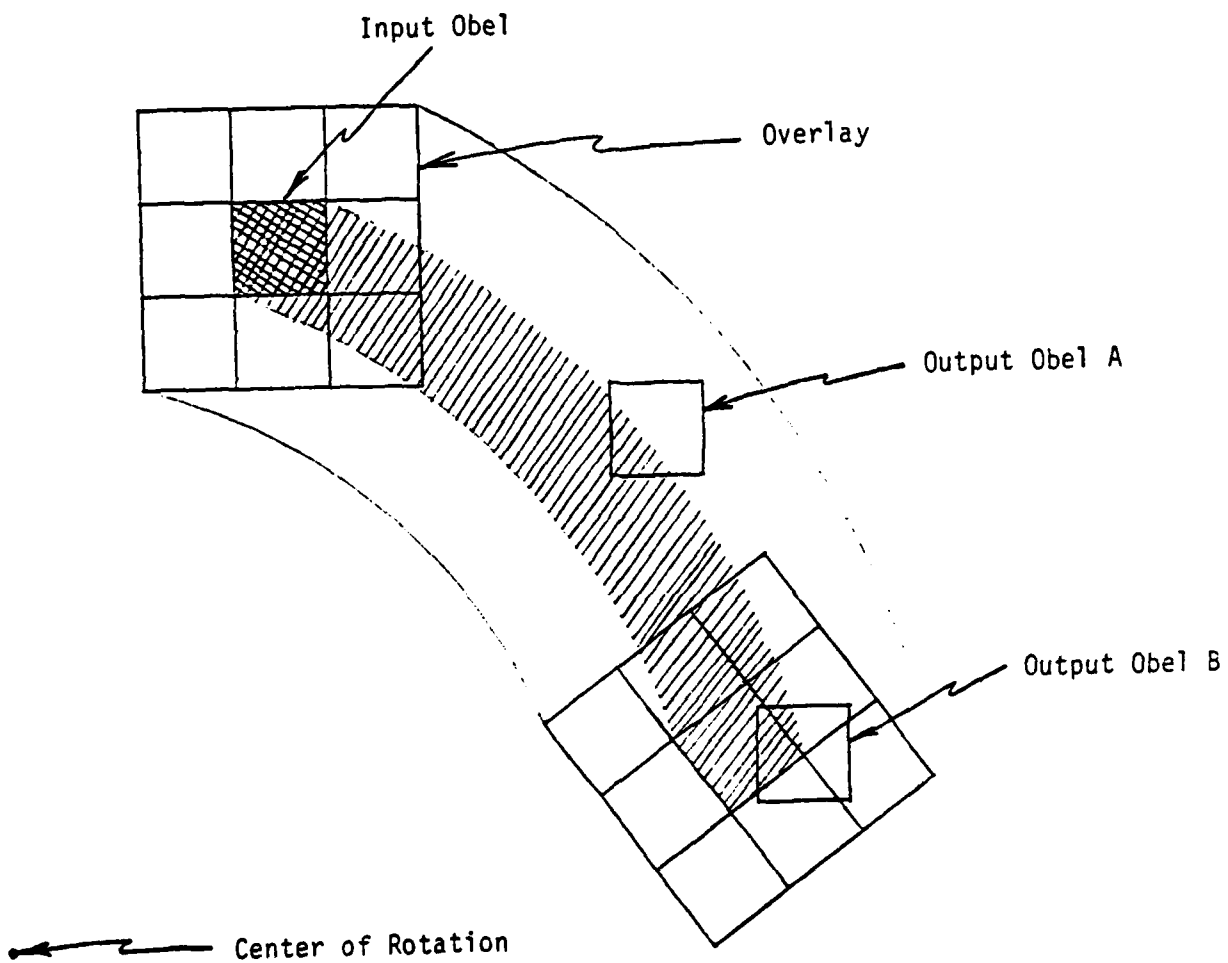
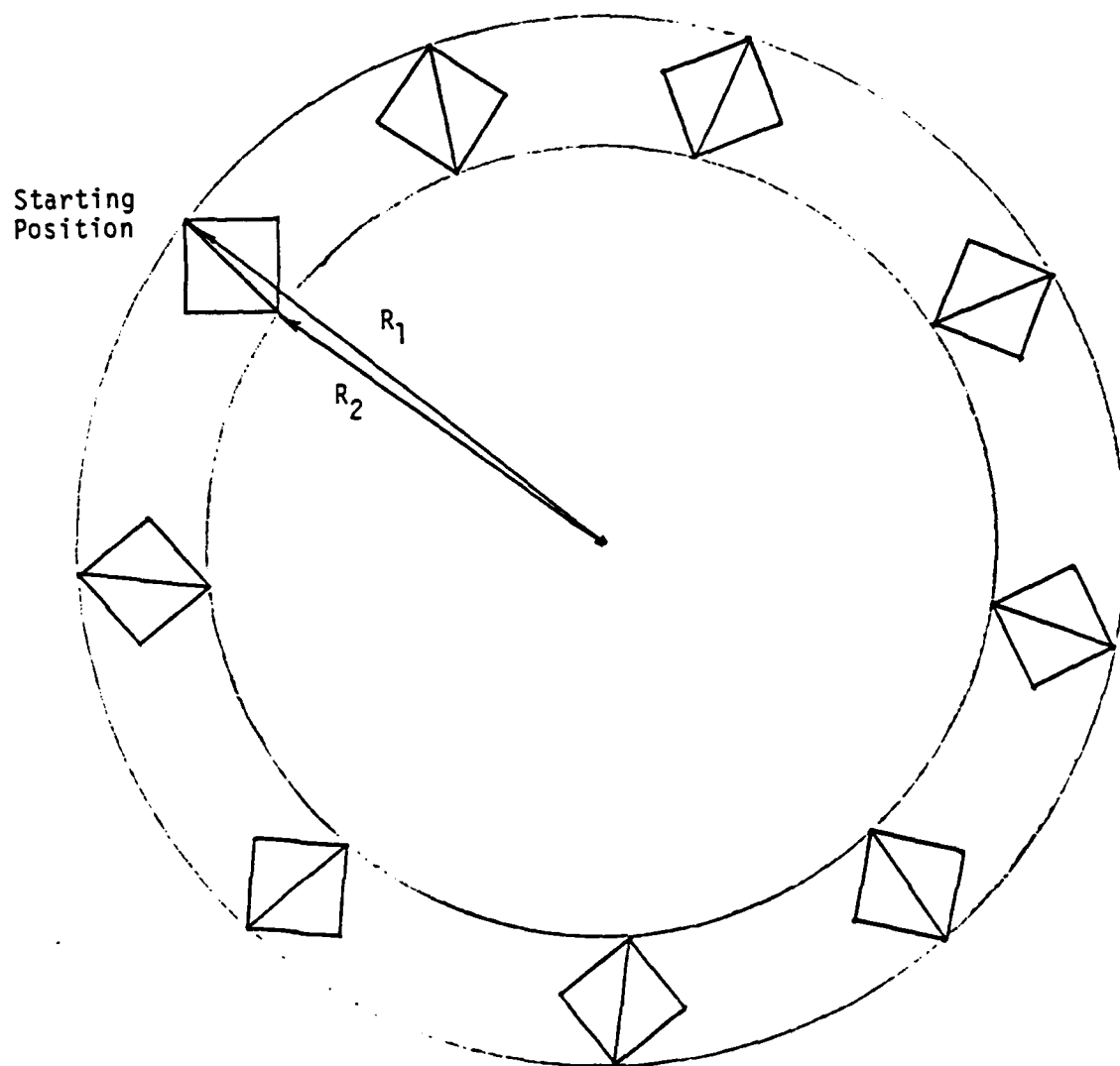
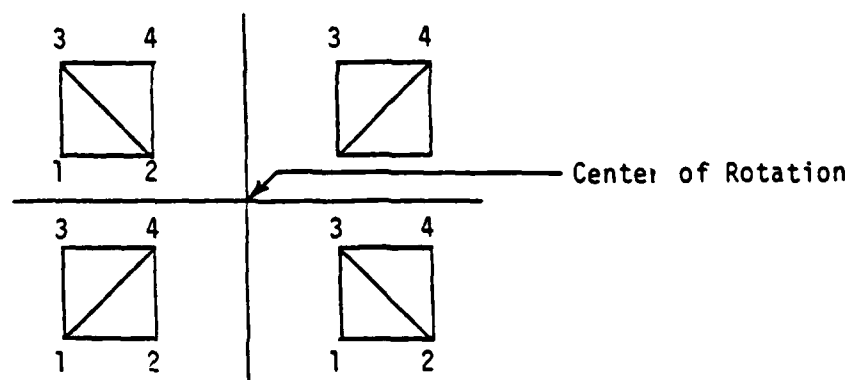


Figure 4.28 Rotational Overlay



(a) Swept Obel



(b) Interior Radius and Exterior Radius Vertex Points vs Starting Quadrant (relative to center of rotation).

Figure 4.29 Obel Sweep

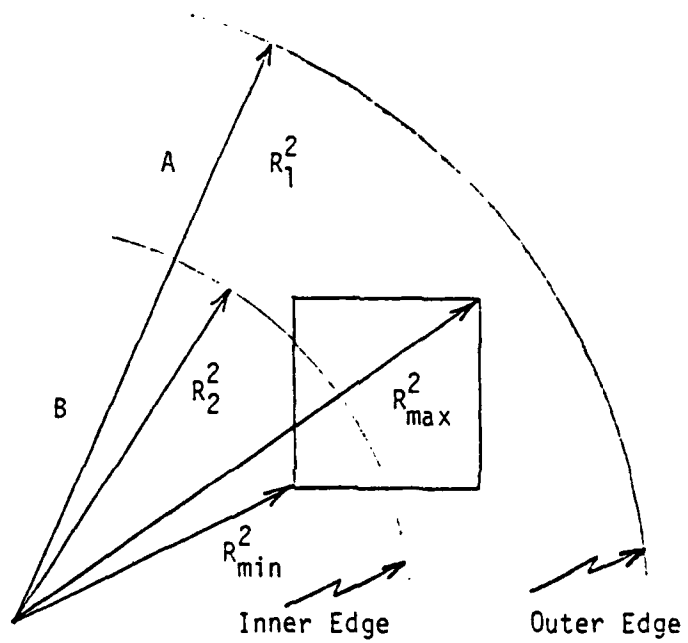


Figure 4.30 Output Obel Intersection Test for Single Band
(No Endpoint Conditions)

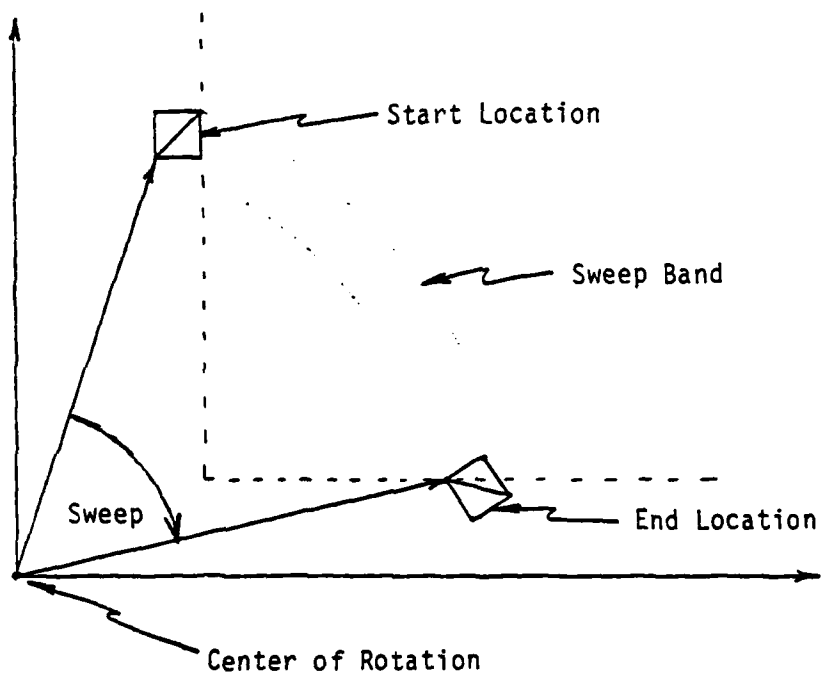


Figure 4.31 Sweep End Condition for a Single Obel

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER IPL-TR-027	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Octree Generation, Analysis and Manipulation		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Donald Meagher		8. CONTRACT OR GRANT NUMBER(s) N00014-82-K-0301
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rensselaer Polytechnic Institute Troy, New York 12181		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 800 North Quincy Street Arlington, VA 22217		12. REPORT DATE April 1982
		13. NUMBER OF PAGES 143
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION, DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) "The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notices hereon." <div style="border: 1px solid black; padding: 5px; display: inline-block;">DISTRIBUTION STATEMENT A Approved for public release; Distribution Unlimited</div>		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) computer graphics solid modeling object manipulation algorithms pattern recognition octree representation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Octree encoding is a solid modeling method designed for the high-speed mani- pulation, analysis and display of arbitrary 3-D objects. The technique is based on a hierarchical 8-ary tree or "octree" for object representation. Octree encoding is presented and analyzed along with a discussion of the major considerations involved in its development. Techniques for the efficient conversion into octrees of convex polyhedra and restricted analytic objects are presented. Strategies for unrestricted and concave object conversion are also discussed. Algorithms for the measurement of object properties (volume, (over)		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

surface area, center of mass, moment of inertia, segmentation of disjoint parts, number of interior voids and a correlation between two objects), geometric operations (translation, scaling, rotation, concatenated geometric operations, nonlinear operations and perspective transformation) and rotational swept volume are developed.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

DATE
ILME